IL BASIC DEL PET E DELL'M20
Paolo Pascolo
e
Carlo Pascolo
CBM
GRUPPO
EDITORIALE
JACKSON

IL BASIC DEL PET E DELL'M20

Paolo Pascolo
e
Carlo Pascolo

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

SOMMARIO

TERZA PARTE

QUARTA PARTE

QUINTA PARTE

SESTA PARTE

INTRODUZIONE

Per descrivere l'organizzazione generale di un calcolatore, il paragone più immediato che viene in mente è la struttura del cervello umano. Ogni nostra azione è subordinata ad un insieme di entità che cercheremo di mettere in relazione con gli elementi costitutivi di un elaboratore elettronico. Le nostre azioni più semplici e immediate sono comandate dal cervello in base a conoscenze ataviche che esistono in noi.

Ad esempio la sequenza di operazioni che compiamo per spostare un braccio quando la mano è vicina ad una fonte di dolore, sono indipendenti dalla nostra volontà e afferiscono ad una zona di memoria con conoscenze primitive.

Operiamo ad un livello diverso quando cerchiamo di compiere operazioni che si ricollegano ad esperienze precedenti. Ad esempio per scrivere una frase ci rivolgiamo ad un'area del cervello in cui sono memorizzate le nostre esperienze in fatto di scrittura (regole grammaticali, utilizzo di una penna, ecc.). Quest'ultima memoria é ben difficilmente scindibile da una terza memoria che contiene i "dati" su cui operiamo. Questa parte conterrà ad esempio le lettere dell'alfabeto che ci serviranno per scrivere o le tabelline aritmetiche che utilizzeremo se desideriamo fare qualche semplice operazione.

Siamo giunti così a descrivere un sistema così costituito:

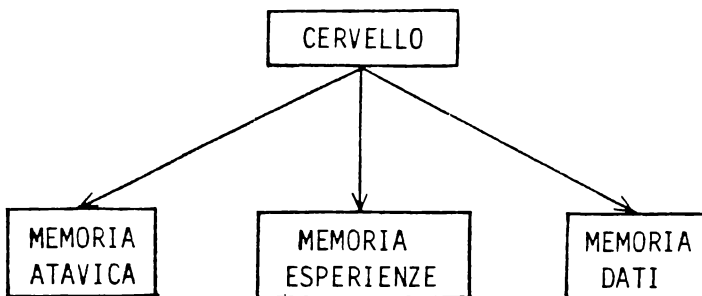



Figura 1.1

Evidentemente ciò non schematizza completamente il sistema "uomo". In aggiunta vi sono dei canali di ingresso, che permettono di acquisire dal mondo esterno gli elementi da elaborare ed inserire nella memoria dati e nella memoria esperienze (tramite la vista, l'udito, il tatto, ecc.).

Analogamente vi saranno dei canali di uscita che permettono il collegamento dal cervello verso l'esterno (tramite ad esempio la parola). Questi due canali verranno chiamati "canali di ingresso-uscita" (I/O, input-output).

Siamo così giunti alla struttura completa:

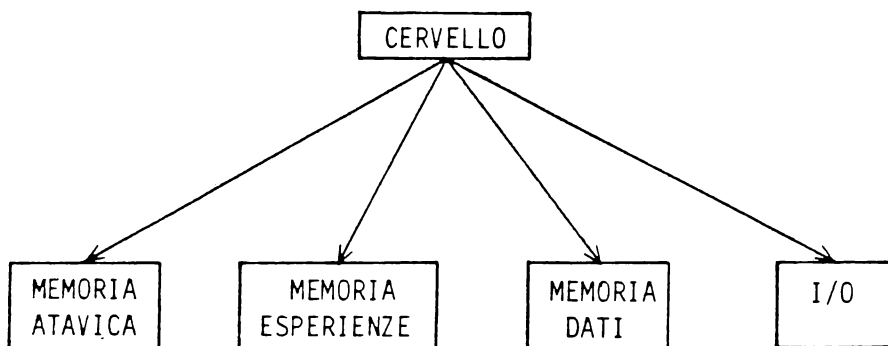



Figura 1.2

Il parallelo a questo punto é completo. Il cervello umano può essere messo in relazione con l'Unità Centrale di Elaborazione; la memoria atavica con il cosiddetto Sistema Operativo, cioé quella parte del sistema che si occupa della gestione al livello più elementare. La memoria esperienze rappresenta il Programma che l'utente scrive, cioè un insieme di operazioni descritte in termini comprensibili all'Unità Centrale. L'area dati contiene gli effettivi dati che vengono passati al calcolatore o che sono frutto di successive elaborazioni. L'Input/Output rappresenta la via attraverso cui il calcolatore colloquia con l'esterno (terminale video, tastiera, stampante, dischi, ecc.).

Si ottiene così la struttura generale del calcolatore che analizzeremo nel prossimo capitolo.

Prima parte

CAPITOLO 1

INTRODUZIONE ALLA PROGRAMMAZIONE

Riprendendo lo schema del calcolatore introdotto nel capitolo precedente (vedi anche Figura 1.3), riconosciamo le seguenti unità:

- unità centrale o processore (P)
- memoria principale (M)
- dispositivi periferici di Ingresso - Uscita (I/O)

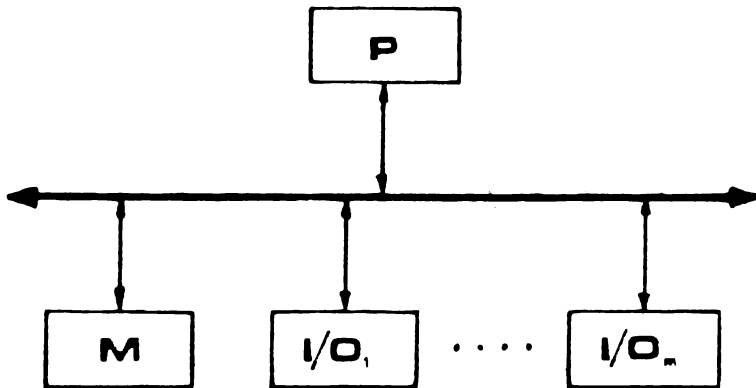



Figura 1.3

Esigenze di semplicità costruttiva hanno eliminato più vie di colloquio con l'Unità Centrale, riducendole ad un unico canale di collegamento (UNIBUS). Le memorie prima definite vengono raggruppate in un'unica unità fisica.

La memoria principale è essenzialmente un magazzino di informazioni, che possono essere sia "dati" che "istruzioni". I dati sono le entità numeriche o alfabetiche elaborate dal processore, o che sono risultate da elaborazioni precedenti. Le istruzioni (statements) sono i comandi che specificano il modo di operare sui dati. L'insieme ordinato delle istruzioni forma il "programma".

In generale, programmi e dati sono memorizzati in zone di memoria differenti, per evitare che il processore interpreti dati come istruzioni e viceversa. Inoltre per istruzioni adibite al controllo del funzionamento della macchina, è preferibile una memorizzazione in speciali memoria a sola lettura (ROM) per avere una protezione da errori di cancellazione o di riscrittura durante il funzionamento.

Il processore, che rappresenta l'unità operativa del sistema, ha il compito di elaborare le istruzioni di un programma, normalmente prelevandole dalla memoria, ed interagendo parzialmente o totalmente con le unità di I/O durante le loro comunicazioni con la memoria principale.

Tipiche funzioni di Processore sono:

- chiamata dell'istruzione
- decodifica
- esecuzione
- analisi di eventuali segnali da parte delle I/O e della memoria

L'insieme delle unità di I/O ha il compito di permettere il colloquio tra il calcolatore e l'ambiente esterno. A seconda del sistema a cui ci si riferisce, queste unità possono comprendere veri e propri processori specializzati verso il controllo di un certo numero di vie di dati da/verso i dispositivi periferici; in ogni caso, saranno sempre presenti delle unità adibite alla gestione delle operazioni tipiche di ogni dispositivo (lettura/scrittura di caratteri o stringhe o records, posizionamento, partenza, stop, ecc.) ed al loro interfacciamento con l'unità centrale. Tipiche unità di I/O sono i terminali, le stampanti, le unità di memorizzazione di massa (dischi, nastri, ecc.).

Tutte le unità appena descritte sono connesse tramite un gruppo di linee di trasmissione che costituiscono il cosiddetto "BUS", cioè una sorta di canale (controllato dal processore), attraverso il quale passano tutte le informazioni in gioco nel sistema.

Nella figura 1.4 abbiamo una schematizzazione del PET 8032 della COM-

MODORE, in cui si possono osservare la tastiera (unità di ingresso), il video (unità di uscita), e il dispositivo di gestione dei dischi (unità di ingresso/uscita), la stampante (unità di uscita) e la memoria principale (unità di ingresso/uscita).

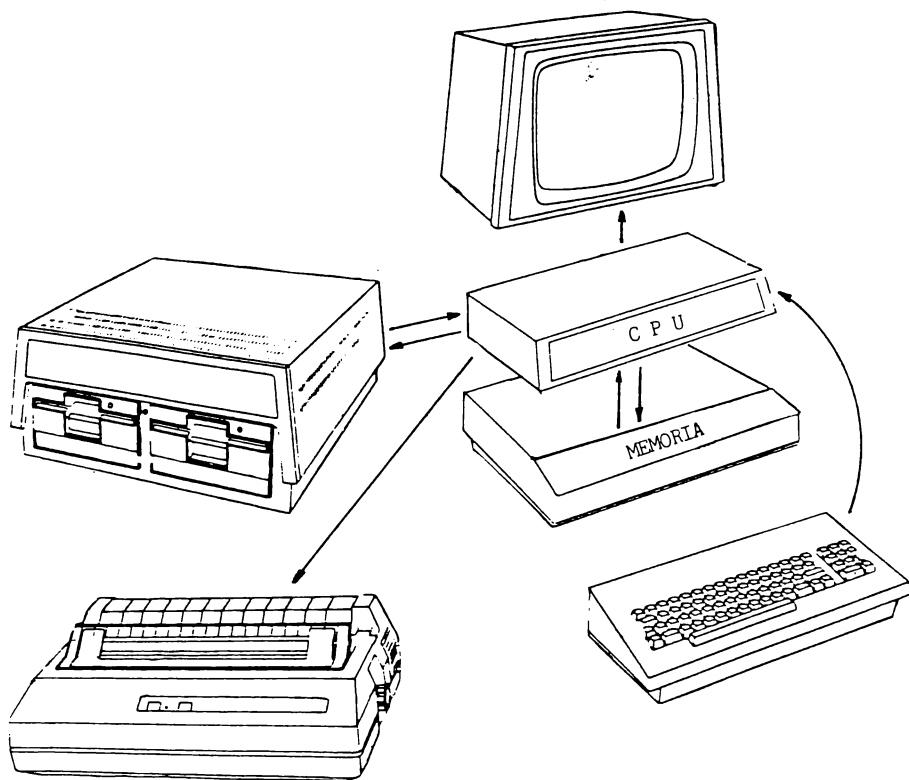



Figura 1.4

CAPITOLO 2

IL SISTEMA BINARIO

Un calcolatore è in grado di riconoscere solamente la presenza o l'assenza di tensione ai capi di un morsetto. È possibile costruire un sistema in cui l'oggetto base è un'entità che assume solamente due stati (sistema binario). In questo sistema, l'unità elementare è il BIT (binary digit), il quale può assumere solamente due valori rappresentati dai simboli 'O' e '1'. Con un bit quindi si può rappresentare il verificarsi o meno di un evento semplice. Per esempio:

— bit=0 lampadina spenta

— bit=1 lampadina accesa

Con due bit si può descrivere un evento composto. Ad esempio:

— bit=0,bit=0 lampadina spenta,di notte

— bit=0,bit=1 lampadina spenta,di giorno

— bit=1,bit=0 lampadina accesa,di notte

— bit=1,bit=1 lampadina accesa,di giorno

Per avere la possibilità di rappresentare eventi di natura più complessa, si definisce il BYTE, cioè un insieme di 8 bit, con cui si può rappresentare fino a 256 aspetti diversi di un evento, tanti quante sono le possibili configurazioni degli 8 bit del byte.

Il byte, come entità singola, viene utilizzato per rappresentare i dati in me-

moria (per esempio le lettere dell'alfabeto, le cifre decimali, i caratteri speciali quali la punteggiatura, ecc.), i numeri in binario ed alcuni comandi standard.

Facciamo qualche esempio. Se si usasse il seguente codice:

il byte 11000001 per la lettera A

il byte 11000010 per la lettera B

il byte 11000011 per la lettera C

la sequenza BAC sarebbe memorizzata con l'espressione:

11000010 11000001 11000011

Si noti che sono necessari 24 simboli binari per rappresentare soltanto 3 lettere dell'alfabeto.

Si opera analogamente per i caratteri che rappresentano le cifre decimali: facciamo bene attenzione a non confondere l'entità astratta numero, con le cifre usate per rappresentarlo: il numero 128 è ben diverso dalla sequenza di cifre "128" (uno due otto). Se supponiamo di codificare il carattere 4 con il byte 11110100 e il carattere 3 con il byte 11110011, la sequenza A4C3 è memorizzata con l'espressione:

11000001 11110100 11000011 11110011

Considerando ora che ogni informazione (dato o istruzione) che viene fornita al calcolatore deve necessariamente essere espressa nel sistema binario, si comprende che risulterebbe estremamente disagevole l'uso del computer (un programma di media grandezza è composto da migliaia di bytes), se il colloquio per l'utente dovesse avvenire esclusivamente in forma binaria.

Un piccolo passo avanti per l'utente si è avuto con l'introduzione del sistema esadecimale, nel quale vengono codificati in un solo simbolo quattro bit, e quindi un byte può essere espresso con soli due simboli esadecimali. Il nome esadecimale deriva dal fatto che sono necessari 16 simboli diversi per codificare le 16 configurazioni possibili dei quattro bit. I 16 simboli usati sono le 10 cifre decimali, più le prime 6 lettere dell'alfabeto, e la codifica è la seguente:

0000 0
0001 1

0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Tornando ai due esempi del sistema binario, ora si avrebbe:

C2C1C3 per la sequenza BAC
C1F4C3F3 per la stringa A4C3

Rimane sottointeso però, che questa notazione è utilizzata solo dall'utente, e che all'interno del computer ci sarà sempre l'espressione in binario.

Un eventuale programma interno penserà a fare le opportune traduzioni dell'esadesimale al binario, e viceversa.

CAPITOLO 3

IL SISTEMA DI NUMERAZIONE BINARIA

Abbiamo già osservato che il computer tratta le cifre numeriche come caratteri qualsiasi, codificandole opportunamente in byte, e la codifica può variare notevolmente da computer a computer.

Ciò non accade per i numeri, i quali sono direttamente rappresentati sfruttando le regole dell'aritmetica binaria. Questa aritmetica è analoga a quella classica decimale, con la differenza che al posto delle dieci cifre (0, 1, 2, ..., 9), se ne usano solamente due: lo 0 e l'1.

Tuttavia i numeri sono interpretati con lo stesso meccanismo. Nel sistema decimale per esempio, le cifre che rappresentano un numero sono disposte da destra verso sinistra secondo potenze crescenti di 10 (10 è la base del sistema decimale):

$$1304 \text{ dec} = 1x10^3 + 3x10^2 + 0x10^1 + 4x10^0$$

Analogamente avviene per i numeri espressi in binario (in cui la base dell'aritmetica è 2):

$$10011 \text{ bin} = 1x2^4 + 0x2^3 + 0x2^2 + 1x2^1 + 1x2^0 = 19 \text{ dec}$$

Facciamo qualche altro esempio di trasformazione di numeri dal binario al decimale:

$$00110100 \text{ bin} = 0x2^7 + 0x2^6 + 1x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0 = 52 \text{ dec}$$

11111111 bin = 255dec

00000000 bin = 0dec

10000000 bin = 128 dec

Per la trasformazione inversa, si opera nel seguente modo: si divide il numero decimale per 2 e si prende il resto di questa divisione come la cifra meno significativa del numero binario risultante; poi si considera il quoziente della divisione e lo si divide ancora per 2: il nuovo resto è la seconda cifra meno significativa del risultato; si considera ora il secondo quoziente e si procede analogamente a quanto già detto, fino ad ottenere come quoziente lo 0 e come resto l'1.

ESEMPIO: si vuole trasformare il numero 57 decimale nel corrispondente numero binario.
Dividiamo 57 per 2:

57:2 = 28 con resto 1

Si ottiene il quoziente 28 e il resto 1: questo è la cifra meno significativa del risultato; dividiamo ora 28 per due:

28:2 = 14 con resto 0

Il 14 è il nuovo quoziente e 0 è la seconda cifra che andrà a formare il numero binario; procedendo nello stesso modo si avrà:

14:2 = 7 con resto 0; 7:2 = 1 con resto 1;
3:2 = 1 con resto 1; 1:2 = 0 con resto 1;

Il resto dell'ultima divisione dà la cifra più significativa del risultato:

57 dec = 111001 bin

Se vogliamo standardizzare ad 8 bit il numero ottenuto, basta aggiungere alcuni zeri a sinistra fino a formare l'ottetto:

57 dec = 00111001 bin

Il numero decimale 57 è quindi rappresentato in binario dal byte 00111001.

Altri esempi:

63 dec = 00111111 bin
32 dec = 00100000 bin
0 dec = 00000000 bin

ESERCIZIO

1. trasformare in binario i seguenti numeri espressi in forma decimale:

   25, 127, 65, 100

2. trovare il valore in base 10 dei seguenti numeri espressi in base 2:

   10110101
   00001111
   01010101
   11111110

3. trovare l'espressione esadecimale dei numeri proposti negli esempi 1. e 2.

CAPITOLO 4

EVOLUZIONE DEI LINGUAGGI

Abbiamo visto che il computer è capace di interpretare solo informazioni codificate opportunamente nel sistema binario. Le regole con cui vengono codificate le informazioni costituiscono un linguaggio per il calcolatore, detto linguaggio macchina o linguaggio direttamente interpretabile dal computer.

Anche le espressioni in esadecimale possono essere considerate un linguaggio macchina, in quanto la traduzione in binario è estremamente semplice.

Un passo avanti nell'uso più razionale delle risorse del computer si è conseguito con l'introduzione dei linguaggi simbolici (chiamati così per differenziarli dai linguaggi macchina precedenti). Un esempio di questo tipo di linguaggi è l'ASSEMBLER.

Questo linguaggio associa ad informazioni specifiche dei nomi simbolici, più espressivi e quindi più facilmente trattabili. Supponiamo che in un particolare codice l'operazione di somma sia espressa con il byte 01101101, che il primo addendo sia 11000100 10110100 (il numero 66740 in binario) ed il secondo sia 00110110 01110010 (il numero 13938).

All'interno del calcolatore l'istruzione di somma per i due addendi potrebbe essere specificata con la terna di informazioni:

01101101
11000100   10110100
00110110   01110010

L'utente, utilizzando il sistema esadecimale avrebbe solo una notevole semplificazione di scrittura/lettura, ma non di leggibilità:

6D C4B4 3672

Se invece si associa al byte della somma, il nome simbolico ADD, al primo addendo il nome ADDENDO1, al secondo il nome ADDENDO2, la notazione precedente diventa:

ADD,ADDENDO1,ADDENDO2

Quest'ultimo modo di scrivere un'istruzione è sufficiente per raggiungere una certa leggibilità dei comandi inviati al calcolatore.

In definitiva, ad ogni istruzione è associato un nome simbolico che ne richiama il significato, e i dati sono trattati più semplicemente associando anche ad essi nomi simbolici. Naturalmente la traduzione dal linguaggio ASSEMBLER al sistema binario è più complessa che dall'esadecimale al binario: tuttavia l'uso di questi linguaggi ha fornito un primo notevole sviluppo dei sistemi di calcolo.

Con i linguaggi di tipo assembler, rimangono però tuttora presenti molte difficoltà di programmazione.

Per esempio questi sono molto legati alla macchina usata, quindi il programmatore deve conoscerne molto bene l'architettura; inoltre è necessario studiare il problema da risolvere in modo molto analitico e dettagliato. Per esempio è necessario controllare ad un livello molto basso (cioè controllando passo per passo ogni più piccola operazione) tutti i dispositivi presenti nel sistema di elaborazione: dalle locazioni di memoria (gli indirizzi dei byte di memoria che sono usati per i programmi e per i dati), ai dispositivi periferici. Infine i programmi non sono trasportabili da un sistema ad un altro e non è possibile programmare in maniera veloce.

Per superare queste limitazioni si è passati ai linguaggi cosiddetti "ad alto livello", dove il programmatore non è tenuto a conoscere la struttura interna della macchina ed è legato soltanto al rispetto delle regole del linguaggio usato. In secondo luogo questo tipo di linguaggi è notevolmente più potente nella risoluzione di problemi che richiedono l'uso del calcolatore. Infine essi sono praticamente indipendenti dai sistemi di elaborazione usati, permettendo quindi il trasporto dei programmi da una macchina ad un'altra.

È appena il caso di aggiungere che ogni programma, scritto in un linguaggio evoluto, dovrà poi essere tradotto (da opportuni programmi interni: i compilatori o gli interpreti) in linguaggio macchina, cioè in un codice binario, prima di poter essere eseguito.

CAPITOLO 5

STRUTTURA BASE DI UN PROGRAMMA

Un generico programma può essere scomposto in segmenti logicamente separabili fra loro e che appartengono alle categorie di MAIN (o programma principale), SUBROUTINE e FUNCTION.

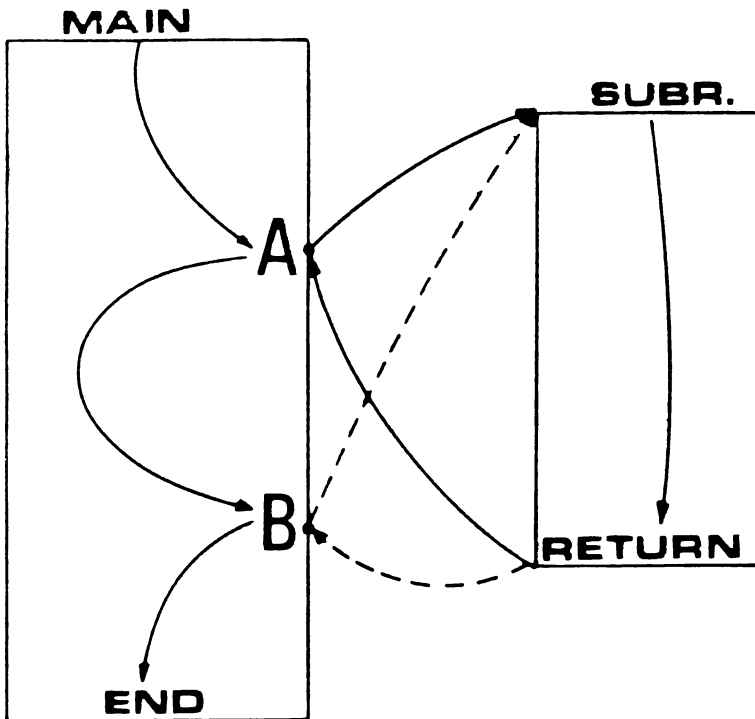



Figura 5.5

In ogni programma esiste un solo MAIN che controlla il flusso dell'intera elaborazione. Nell'ambito del MAIN ci possono essere delle interruzioni nella sequenzialità di esecuzione delle istruzioni, con opportuni salti a particolari sottoprogrammi. Le FUNCTIONS e le SUBROUTINES sono utilizzate per svolgere certe funzioni o certe elaborazioni, che possono rendersi necessarie una o più volte nel corso di esecuzione del programma principale. In Figura 5.5 è rappresentato il flusso di esecuzione del programma principale e di una SUBROUTINE.

Nei punti A e B avviene l'interruzione del programma MAIN, con conseguente salto al sottoprogramma SUBROUTINE. Una volta che la SUBROUTINE é stata eseguita, il controllo ritorna al MAIN per mezzo di un "salto di ritorno".

Una SUBROUTINE può, a sua volta, chiamare un'altra SUBROUTINE; e questa chiamarne un'altra ancora, e così di seguito. Questo tipo di struttura é detto annidamento. In generale i sistemi di elaborazione pongono un limite massimo al numero di SUBROUTINE annidate.

L'impiego di sottoprogrammi permette, oltre ad un certo risparmio di lunghezza del programma, anche una maggiore organizzazione logica (strutturazione) del flusso di esecuzione delle istruzioni, con una conseguente maggiore facilità di lettura e di interpretazione dei programmi nel loro complesso.

Una FUNCTION é concettualmente simile ad una SUBROUTINE. L'unica differenza consiste nel fatto che, la prima restituisce un unico valore calcolato, mentre la seconda esegue esclusivamente un flusso logico di programma.

Un esempio di FUNCTION può essere la funzione SQR del BASIC. Questa funzione serve per calcolare la radice quadrata di un numero:

```
Y=SQR(X)
```

Nella istruzione BASIC appena scritta, la funzione SQR calcola la radice del numero rappresentato dalla variabile X e restituisce tale valore, assegnandolo alla variabile Y.

CAPITOLO 6

COSTANTI E VARIABILI DI UN PROGRAMMA

Sono le entità base su cui opera ogni programma.

I dati di tipo costante non cambiano il loro valore per tutto il tempo di vita del programma. Nonostante piccole differenze da calcolatore a calcolatore, ne esistono sostanzialmente di due tipi: numeriche e simboliche (alfanumeriche).

I dati di tipo variabile sono dei nomi con cui vengono indicati opportuni campi di memoria. In queste aree sono contenuti i valori che si riferiscono ai nomi suddetti. Il valore di una variabile le può essere assegnato direttamente dal programmatore, oppure può essere il risultato di calcoli compiuti dal programma. Per esempio se scriviamo:

```
B=3
```

l'area di memoria riservata alla variabile B assume il valore numerico 3. Se poi scriviamo:

```
A=B+2
```

l'area di memoria riservata alla nuova variabile A assume un valore pari alla somma di quello relativo alla variabile B e della costante numerica 2. In definitiva quindi nell'area di memoria assegnata ad A viene immagazzinato il valore numerico 5.

Naturalmente é possibile anche la seguente notazione:

```
A=A+1
```

la quale sta a significare che il nuovo valore assegnato ad A é quello precedente incrementato di 1.

Si noti che il simbolo uguale '=' ha un significato diverso da quello normalmente assegnato, cioé di uguaglianza della parte destra con la parte sinistra del simbolo. Esso deve qui intendersi come un aggiornamento dell'area di memoria riservata alla variabile specifica a sinistra di tale simbolo.

Seconda parte

CAPITOLO 1

IL LINGUAGGIO BASIC

LE COSTANTI

Le COSTANTI DI TIPO STRINGA (o simboliche), sono costituite da una successione di caratteri quali lettere dell'alfabeto, cifre e ogni altro simbolo presente nella tastiera racchiuso tra virgolette.

L'intera stringa può contenere al massimo 255 caratteri.

ESEMPIO:

corretti	**sbagliati**
"ESEMPIO 12"	LUN
"12.15"	12.152"
"SABATO"	"MARZO"
"P#.3 & ASD+<>'$%3.465"	

Le COSTANTI NUMERICHE rappresentano dei valori esclusivamente numerici. Possono essere positive o negative, intere o con parte intera e parte decimale. Per separare la parte intera da quella decimale si usa, seguendo la notazione anglosassone, il punto anziché la virgola.

Le costanti intere sono un qualsiasi numero intero compreso tra —32768 e +32767. Non hanno il punto decimale. Il "range" (l'intervallo) entro cui possono assumere valori tali costanti, dipende dalla rappresentazione interna dei numeri interi. Utilizzando 16 bit si possono specificare, al massimo, 2 elevato alla 16 combinazioni differenti; tante quanti sono i numeri tra —32768 e +32767.

ESEMPIO:

corretti	sbagliati	
12	12.1	(non è intero)
–128	–32769	(fuori range)
0	32800	(fuori range)

Le costanti non intere o in virgola mobile (floating point) possono essere di due tipi: semplice o doppia precisione (quest'ultima solo nel computer M20).

Le costanti in semplice precisione possiedono al massimo 9 cifre significative nel COMMODORE e 5 nell'M20, e sono espresse in una delle seguenti forme:

a) contengono il punto decimale

12.3
– 7.0011
0.0

b) sono maggiori di 32767 o minori di –32768

32820
100000
–999999

c) sono espresse in forma esponenziale.

235E6 equivale a 2.35×10^{8}
– 28E–7 equivale a -2.8×10^{-6}

La forma esponenziale é simile alla notazione scientifica, ove il termine 10 è sostituito dalla lettera E.

Il "range" di variabilità dell'esponente, dipende dal tipo di calcolatore e dal numero di bit che esso utilizza per rappresentarlo. Nei calcolatori COMMODORE, l'esponente può variare tra –38 e +38.

ESEMPIO:

corretti	sbagliati	
145.2	145,2	(la virgola)
– 18.0E12	12E39	(troppo grande)
256E–07	7E–51	(troppo piccolo)

Nel computer M20, oltre ai precedenti tipi di costanti numeriche, si possono definire le costanti in 'doppia precisione' in cui ci sono 15 cifre significative e l'esponente può variare tra −308 e +308. Per esprimere una costante in doppia precisione si utilizza la forma esponenziale sostituendo la lettera 'D' alla lettera 'E':

235.12D150
125.123654025D−52

Il massimo numero esprimibile in questa forma è
1.797693131486231D+308.

È inoltre possibile esprimere costanti in doppia precisione posponendo al numero il carattere '#':

125#
32.1458#

CAPITOLO 2

LE VARIABILI

I nomi delle variabili BASIC nel COMMODORE possono avere una qualsiasi lunghezza, ma solo i primi due caratteri sono significativi (per esempio i nomi VERDE e VENDITE sono considerati uguali). Nei nomi si possono usare anche numeri, ma il primo carattere deve essere una lettera.

ESEMPIO:

A1
TIPO$
AA23FG

I nomi di variabili non possono essere nè contenere parole riservate, cioè tutti i comandi BASIC, le istruzioni, i nomi di funzione e di operatori funzionali.

A seconda della loro configurazione, i nomi di variabili rappresentano valori numerici interi o decimali, oppure stringhe di caratteri. I nomi di stringhe devono terminare con il simbolo $, i nomi di variabili numeriche intere devono terminare con %.

ESEMPIO:

NUM%	numero intero
SOMMA	numero decimale
MOLT	numero decimale
PAROLA$	stringa

Sebbene negli esempi precedenti abbiamo usato nomi di variabili lunghi più di due caratteri, è preferibile definire variabili di non più di due caratteri,

per evitare errori dovuti all'uso di nomi diversi che, in realtà, si riferiscono alla stessa variabile: le variabili A13 e A15 sono interpretate come un'unica variabile!!

Nel Computer M20 i nomi delle variabili possono essere lunghi fino a 40 caratteri, non possono essere parole riservate ma possono contenerle. Per esempio la parola "OR" é riservata (come vedremo in seguito): in entrambi i computer non é possibile utilizzare OR come variabile, e solo nell'M20 è possibile utilizzare la variabile ORA.

Per definire variabili che conterranno valori in doppia precisione (solo M20), si deve aggiungere la desinenza #.

Per assegnare un certo valore ad una variabile è sufficiente scrivere il simbolo = seguito dall'espressione desiderata, per esempio:

```
A1=34.2
A3$="ESEMPIO"
DF=12*3+56.2/88
DF%=12
HB=56
```

Per controllare una certa assegnazione, si può operare nel seguente modo: si digita sulla tastiera l'istruzione di assegnamento (ricordando di premere il tasto 'RETURN' (nell'M20 è il tasto a forma di L rovesciata) alla fine dell'espressione), poi si scrive il comando PRINT seguito dal nome della variabile e si batte nuovamente il tasto 'RETURN'. Sullo schermo comparirà il valore attuale della variabile.

ESEMPIO:

DF$="PROVA" 'RETURN'	(a questo punto la variabile DF$ assume il valore PROVA)
PRINT DF$ 'RETURN'	(si visualizza la parola PROVA)

la parola PRINT può essere sostituita dal simbolo '?'. Proviamo infatti a scrivere:

```
DF$="CO.R.EL. F." 'RETURN'
? DF$ 'RETURN'
```

si visualizzerà la sigla CO.R.EL. F.

Possiamo visualizzare anche variabili numeriche (ci si deve sempre ricordare di premere il tasto 'RETURN' alla fine di ogni comando o di una istruzione):

```
AA=123.45
BB%=12
CC$="STRINGA"
DF=145
? AA,BB%,CC$,DF,DF$
```

sul video compariranno su una stessa riga:

```
123.45    12    STRINGA    145    CO.R.EL. F.
```

Facciamo ora qualche esempio di assegnazione sbagliata.

```
F%=100000
```

La risposta a questa assegnazione è:

? ILLEGAL QUANTITY ERROR	nel COMMODORE
OVERFLOW	nell'M20

Illegal quantity naturalmente significa quantità illegale; overflow significa oltre misura: infatti F% è una variabile intera, che quindi non può contenere numeri maggiori di 32767 o minori di –32768. Si deve perciò scrivere

```
F=100000
```

in cui F è una variabile floating point.

Altri esempi di assegnazioni non corrette:

HH="B"	HH è numerica (corr.: HH$="B")
X=3 x 10^1	si deve scrivere X=3E1
J=1.5E50	il numero è troppo grande
1C=12.5	il primo carattere del nome non è alfabetico (corr.: C1=12.5)
A$=3	A$ non è una var. numerica: (corr.: A$="3" oppure A=3)

A$=A	mancano gli apici: (corr.:A$="A")
A=$="BUNNY"	il carattere = non è alfanumerico (corr.: A3$="BUNNY")

ESERCIZIO

1. trovare gli errori nelle seguenti assegnazioni:

1) FF$=R3
2) X%4="ABC"
3) FP=.25E-39
4) 2H%=7
5) J%="FFF"
6) HZ=27E40
7) F7%=-40123
8) EP%=43-
9) MN$=44
10) DN=2700E36
11) SS%=-30.01
12) $A="TIPO"

CAPITOLO 3

VARIABILI ARRAY (VETTORI E MATRICI)

Un array è un gruppo (o tabella) di valori associati ad un unico nome. Ogni elemento dell'array è individuato tramite il nome dell'array e il numero d'ordine dell'elemento, nell'array stesso. Per esempio si esamini l'array di 7 elementi così composto:

1:	"LUN
2:	"MAR"
3:	"MER"
4:	"GIO"
5:	"VEN"
6:	"SAB"
7:	"DOM"

Si denoti con GN$ l'intero array di 7 elementi: con GN$(6) ci si riferisce a "SAB".

Gli array possono avere più dimensioni, per esempio la tabella pitagorica potrebbe essere memorizzata con l'array PT% a due dimensioni:

	1:	2:	3:	4:	5:	6:
1:	1	2	3	4	5	6
2:	2	4	6	8	10	12
3:	3	6	9	12	15	18
4:	4	8	12	16	20	24

Ovviamente con PT%(4,3) ci si riferisce al numero intero 12.
Si possono definire array di dimensioni maggiori a 2.
Un esempio di array a quattro dimensioni può essere il seguente:

T(C,G,M,O)

con

C = città della regione Friuli Venezia Giulia
(1,...,4)

G = giorno di un particolare mese dell'anno
(1,...,31)

M = mese
(1,...,12)

O = massima/minima
(1,2)

e con l'array T si vuole indicare la temperatura massima e minima giornaliera.

Se si fa l'assegnazione

TS = 1
UD = 2
GO = 3
PN = 4

si possono registrare le varie temperature delle città del Friuli.

Ad esempio scrivendo

T(2,30,12,1) = 2

si memorizza la temperatura massima della città di Udine, del giorno 30 dicembre.

Questo array occupa un'area di memoria pari a 2976=(4*31*12*2) campi.

CAPITOLO 4

ESPRESSIONI ED OPERATORI

Un'espressione è, in generale, una combinazione di dati numerici o stringhe, fatta per mezzo di operatori. Come caso particolare, una espressione può essere composta da una sola costante o da una variabile, oppure da una stringa.

Gli operatori eseguono operazioni logico-matematiche sui dati. Nel BASIC sono previste quattro categorie di operatori.

aritmetici
relazionali
logici
funzionali

OPERATORI ARITMETICI

Gli operatori aritmetici sono (in ordine di priorità):

operatore	operazione	esempio
↑	esponenziazione	X↑Y
+,—unari (prefissi)	afferm., neg.	+X, —X
*,/	molt., div.	X*Y, X/Y
+,—binari (infissi)	somma, sottrazione	X+Y, X—Y

Per cambiare l'ordine nel quale vengono eseguite le operazioni, si usano le parentesi: le operazioni dentro le parentesi sono eseguite per prime.

ESEMPI:

A=X+Y*2	corrisponde a	A=Y+2Y
A=X–Y/Z	corrisponde a	A=X–(Y/Z)
A=X*(–Y)	corrisponde a	A=X(–Y)
A=6*X↑2+3*X–4	corrisponde a	$A=6X^2+3X-4$

L'espressione valutata viene assegnata alla variabile a sinistra dell'uguale.

Se durante la valutazione di una espressione avviene una divisione per 0, appare sul video l'espressione: "DIVISION BY ZERO ERROR". Se avviene che il risultato di una espressione sia più grande del massimo possibile, sul video appare: "OVERFLOW ERROR".

ESEMPIO

1. scriviamo in termini BASIC l'area del triangolo e del trapezio:

TR=B*H/2

AT=(B1+B2)*H/2

Nell'M20, oltre agli operatori precedenti, sono disponibili anche i seguenti:

\	divisione intera
MOD	modulo

In entrambi i casi gli operandi sono approssimati all'intero più vicino. Nella divisione intera il risultato è troncato all'intero; nell'operazione di modulo il risultato è il resto ottenuto con una divisione intera.

La priorità di questi operatori è compresa tra gli operatori *,/ e gli operatori +,– binari.

ESEMPIO

10 \ 4 = 2 (divisione intera)
9 MOD 4 =1 (op. modulo)

ESERCIZIO

2. scrivere in termini BASIC le espressioni risolutive dei seguenti problemi:

1) volume della sfera: $VS = \frac{4\pi}{3} R^3$

2) montante: $MT = 100000 \left(1 + \frac{I}{100}\right)$

3) radici di un'equazione di secondo grado: $X = \frac{B \pm \sqrt{B^2 - 4AC}}{2A}$

4) volume del cono: $VC = \frac{\pi R^2 H}{3}$

5) imponibile: $IM = \frac{TM}{1 + \frac{A}{100}}$

6) I.V.A.: $IV = TM \left(\frac{1 - TM}{1 + \frac{A}{100}} \right)$

7) freccia di una trave rettangolare incastrata ad un estremo:

$$F = \frac{PL^3}{3E\left(\frac{BH^3}{12}\right)}$$

OPERATORI RELAZIONALI

Gli operatori relazionali sono usati per confrontare due valori. Il risultato del confronto può essere "TRUE" o "FALSE", (rispettivamente VERO o FALSO). La codifica interna del valore TRUE son due bytes di '1' (che corrisponde al valore —1 espresso in una opportuna codifica interna); analogamente la codifica del FALSE sono due bytes di '0' (corrispondente al valore 0). Questi operatori relazionali sono principalmente usati nella istruzione IF che si vedrà in seguito.

operatori	**test**
=	uguaglianza
< >	disuguaglianza
<	minore a
>	maggiore a
< =	minore o uguale a
> =	maggiore o uguale a

(NB: Il segno = è usato anche per assegnare un valore ad una variabile).
Quando operatori aritmetici e relazionali sono combinati in una espressione, gli aritmetici sono sempre eseguiti per primi:

X+Y<(T−1)/Z

è vera se X+Y è minore di T−1 diviso per Z.

Un'espressione relazionale può essere assegnata ad una variabile numerica, e il risultato sarà il valore 0 se l'espressione è falsa, il valore −1 se invece è vera.

ESEMPIO:

VL=5<8, dopo questo assegnamento la variabile VL contiene il numero −1.

ESERCIZIO

3. nell'ipotesi in cui A=5 e B=3 determinare il valore della variabile VL nei seguenti casi:

1) VL=A<B
2) VL=B=A
3) VL=B< >A
4) VL=3
5) VL=(B<=A)
6) VL=(A<A+B)

OPERATORI LOGICI

Gli operatori logici permettono test su relazioni multiple, manipolazioni di bit, operazioni Booleane. Questi operatori ritornano un valore che è TRUE (vero:−1) o FALSE (falso:0). In una espressione, gli operatori logici sono eseguiti dopo gli operatori aritmetici e relazionali. Gli operatori logici sono applicabili sia ad espressioni aritmetiche che relazionali. Il risultato di una operazione logica è determinato come mostrato in seguito; X ed Y sono due espressioni di qualunque genere e gli operatori sono descritti in ordine di priorità decrescente:

NOT:

X	NOT X
falso	vero
vero	falso

(cioè se l'espressione X è vera, NOT X è falsa, e viceversa)

AND:

X	Y	X AND Y
falso	falso	falso
falso	vero	falso
vero	falso	falso
vero	vero	vero

(cioè X AND Y è vera soltanto se sia X che Y sono veri)

OR:

X	Y	X OR Y
falso	falso	falso
falso	vero	vero
vero	falso	vero
vero	vero	vero

(cioè X OR Y è vera se almeno uno tra X e Y sono veri).

Analogamente agli operatori relazionali, gli operatori logici possono essere usati per operare decisioni riguardanti il flusso del programma. In genere essi sono usati per connettere due o più relazioni.

ESEMPIO:

    D<200 AND f<4

è vera solo se entrambe le relazioni sono vere

    D<200 OR F<4

è vera se almeno una relazione è vera

    NOT K<=0

è vera solo se K>0.

ESEMPIO

2. sia A=0 e B=−1:

   VL=A AND B dà come risultato VL=0
   VL=A OR B dà come risultato VL=−1
   VL=NOT A dà come risultato VL=−1
   VL=NOT B dà come risultato VL=0
   VL=A OR B AND NOT B dà come risultato VL=0
   quest'ultima è equivalente a VL= A OR (B AND (not b))

Nell'M20, sono disponibili altri tre operatori logici, di priorità inferiore a OR:

XOR:	X	Y	X XOR Y
	falso	falso	falso
	falso	vero	vero
	vero	falso	vero
	vero	vero	falso

(cioè X XOR Y è vera soltanto se X e Y hanno valori diversi).

EQV:	X	Y	X EQV Y
	falso	falso	vero
	falso	vero	falso
	vero	falso	falso
	vero	vero	vero

(cioè X XOR Y è vera soltanto se X e Y hanno valori uguali).

IMP:	X	Y	X IMP Y
	falso	falso	vero
	falso	vero	falso
	vero	falso	vero
	vero	vero	vero

(cioè X IMP Y è falsa soltanto se X è falso e Y è vero).

4. sia A=0, B<>0, C=0, D<>0, determinare il valore della variabile VL dopo le seguenti assegnazioni:

1) VL=C OR D
2) VL=A AND D
3) VL=NOT D
4) VL=A AND B OR D
5) VL=NOT (A AND B) AND NOT (B OR C)
6) VL=A OR B AND C OR D
7) VL=(5<4) OR (7<3)
8) VL=((5<4) OR (7>3)) AND (4>2)
9) VL=NOT (7+2>8 AND 5>7*(−1))
10) VL=(3>5 OR 6<9) AND NOT (2>8)

OPERATORI FUNZIONALI

Un operatore funzionale è usato in una espressione, per richiamare una operazione predefinita, che viene eseguita su un operando. Il COMMODORE-BASIC ha alcune funzioni intrinseche residenti nel sistema, come SQR (radice quadrata), SIN (funzione seno), ecc. Altre funzioni, di interesse particolare, possono essere definite, come vedremo in seguito, dal programmatore, a seconda delle necessità di calcolo.

CAPITOLO 5

MODI DI OPERAZIONE DEL BASIC

Nel "modo diretto", i comandi e le istruzioni BASIC, non sono preceduti dai numeri di linea (etichette associate ad ogni linea di istruzione), e sono eseguiti immediatamente. I risultati di operazioni logiche e aritmetiche possono essere visualizzati ed immagazzinati per un ulteriore uso. Questo modo è usato essenzialmente per sfruttare l'elaboratore come calcolatore da tavolo, cioè per ottenere veloci calcoli che non richiedono programmi, e per il "debugging".

Il debugging è un lavoro di tipo "investigativo" che serve a correggere errori nascosti nel programma. A volte infatti, può capitare che, corretti gli errori grammaticalli (scorrettezze nella forma delle istruzioni), il programma vada in esecuzione, ma non dia risultati esatti. Gli errori presenti sono da ricercare nella costruzione logica del programma. In generale gli errori logici sono difficili da scoprire, e richiedono un ripasso accurato dell'effettivo lavoro svolto da ogni singola istruzione (o da gruppi di istruzioni) e dal flusso effettivo del programma.

Un esempio: si provi a digitare sul computer la seguente espressione:

```
PRINT 6*8-2
```

e si premi il tasto 'RETURN'. L'elaboratore calcola l'espressione e visualizza il risultato; sul video compare:

```
46
READY.
```

e la macchina si predispone ad eseguire altri comandi.

Il "modo indiretto" è usato per introdurre programmi. Ogni linea di pro-

gramma è preceduta da un numero, ed è memorizzata all'interno del calcolatore, in maniera sequenziale. Il programma è poi eseguito digitando il comando RUN.

FORMATO DI LINEA

nnnnn /istruzione BASIC/ (* :/istruzione BASIC/ *)...'RETURN'

Se si definiscono su una stessa linea (che può essere lunga al massimo 80 caratteri nel COMMODORE, e 256 nell'M20), più istruzioni BASIC, queste devono essere disgiunte da ':'. Il simbolo 'RETURN' indica il tasto RETURN.

D'ora in avanti, per la definizione della sintassi delle istruzioni varranno le seguenti regole:

1. le parole in maiuscolo devono essere digitate come mostrato
2. le parole fra le barre inclinate// devono essere sostituite dal programmatore
3. ciò che appare fra (* e *) è opzionale
4. tutta la punteggiatura, eccetto (* *) / deve essere digitata come mostrato
5. i tre puntini ... indicano la possibilità di ripetizione
6. il punto esclamativo ! indica espressioni mutualmente esclusive (bisogna sceglierne una).

NUMERI DI LINEA

Ogni linea di programma BASIC inizia con un numero di linea. I numeri di linea indicano l'ordine con cui le linee di programma vengono memorizzate (e quindi eseguite). Essi devono appartenere al range 0, 63999 per il COMMODORE e al range 0, 65529 per l'M20.

ESEMPIO

3. A questo punto proviamo a scrivere e ad eseguire un programma (valido per entrambe le macchine), che calcola una semplice espressione matematica:

$$R=B^2-4AC$$

Il programma deve quindi assegnare dei valori alle variabili A, B, C, calcolare l'espressione ed assegnare il risultato a R. Inoltre faremo visualizzare sul video il valore di R. Il programma è:

```
10  REM PROG1.1:CALCOLO DI UNA ESPRESSIONE
20  A=3
30  B=4
40  C=5
50  R=B↑2 – 4*A*C
60  PRINT "R="R
```

L'istruzione etichettata da 10 è un commento. Le istruzioni etichettate dai numeri 20,30,40, assegnano alle variabili a sinistra del simbolo '=', i numeri specificati a destra.
L'istruzione 50 calcola l'espressione ed assegna ad R il risultato. L'istruzione 60 stampa la stringa "R=" ed accanto a questa, il valore attuale di R. (Tutte le istruzioni utilizzate saranno dettagliatamente spiegate nel seguito).

Le istruzioni 20, 30, 40, possono essere raccolte in un'unica linea di programma, separandole con il simbolo ':':

```
10  REM PROG1.1BIS:CALCOLO DI UNA ESPRESSIONE
20  A=3:B=4:C=5
30  R=B↑2–4*A*C
40  PRINT"R="R
```

Per eseguire il programma operiamo come segue.
Digitiamo il comando NEW, che cancella la memoria: questo comando è solitamente usato quando si vuole introdurre un nuovo programma da tastiera.
Scriviamo ora il programma, facendo attenzione alla sequenzialità delle istruzioni ed alla loro correttezza formale. Se durante la digitazione viene introdotto un carattere sbagliato, lo si può cancellare usando il tasto 'DEL', e riscrivendo il carattere desiderato.

Sull'M20 un carattere sbagliato viene cancellato premendo i tasti "control" H.

Introdotto il programma, si digiti il comando RUN per l'esecuzione. Il risultato deve essere:

```
R=–44
```

La sintassi completa dei comandi RUN e NEW sarà vista nei prossimi capitoli.

CAPITOLO 6

I COMANDI BASIC NEW, RUN, LIST

comando NEW

NEW

È usato per cancellare il programma e tutte le variabili in memoria.

Questo comando è digitato nel modo diretto, prima dell'introduzione di un nuovo programma.

La forma è identica per entrambi i computer.

comando RUN

COMMODORE: **RUN (* /numero di linea/ *)**

Esegue il programma in memoria. Se è specificato il numero di linea (che è opzionale), l'esecuzione inizia da quella linea; altrimenti l'esecuzione inizia dalla linea con il numero più basso.

M20: **RUN (* /num. linea/!/nome programma/ (* ,R *) *)**

Se si utilizza la forma RUN oppure RUN /num. linea/ la funzione del comando è identica al COMMODORE. Se si specifica RUN /nome.programma/ l'M20 carica dal disco il programma specificato da /nome programma/ e lo mette in esecuzione. Se inoltre si aggiunge la desinenza ',R' (che può essere utilizzata solo con quest'ultima forma di RUN), i files (archivi dati su disco) eventualmente aperti da programmi eseguiti precedentemente, non verranno chiusi.

ESEMPIO

4. consideriamo il seguente programma (valido per entrambe le macchine):

```
10  A=1000
20  PRINT"A="A
30  B=2000
40  PRINT"B="B
```

Se lo eseguiamo con RUN, sullo schermo compare:

```
A=1000
B=2000
```

Se lo eseguissimo invece con RUN 30, comparirebbe la scritta:

```
B=2000
```

in quanto non verrebbero eseguite le prime due istruzioni.

Se sull'M20 scriviamo:

```
RUN "1:prova"
```

il programma 'prova' residente sul drive 1, è caricato in memoria e quindi messo in esecuzione.

comando LIST

Questo comando esiste in due formati per il COMMODORE:

LIST (* /numero di linea/ *)

LIST (* /numero di linea/ *) – (* /numero di linea/ *)

Serve per visualizzare tutte o parte delle istruzioni del programma attualmente in memoria.

Nel primo formato, se è omesso il numero di linea, il programma è listato completamente. Per interrompere lo scorrimento delle linee di programma si prema il tasto 'STOP' ('Control' C sull'M20). Se è incluso anche il numero di linea, viene visualizzata solo la linea specificata.

Nel secondo formato sono presenti le seguenti opzioni:

a) se è specificato solo il primo numero, la lista comprende le linee da quel numero in poi.

b) se è specificato solo il secondo numero, la lista comprende le linee dall'inizio fino a quel numero.

c) se sono specificati entrambi, la lista comprende il range tra i due numeri.

ESEMPIO

5. consideriamo il programma PROG1.1 dell'esempio 3.

LIST :visualizza il programma in memoria
LIST 50:visualizza la linea 50
LIST 30—:tutte le linee da 30 alla fine
LIST —40:dall'inizio a 40
LIST 20—50:da 20 a 50

Nell'M20 sono presenti tutte le precedenti possibilità inoltre esistono i formati:

LIST .
LIST .—/num. linea/

Queste istruzioni visualizzano l'istruzione corrente o l'intervallo di istruzioni comprese tra l'istruzione corrente e quella specificata con /num. linea/.

Sul computer COMMODORE, il comando LIST sta alla base del servizio di "editing", che consente di effettuare in modo agevole, aggiornamenti e correzioni sui programmi.
Visualizzato il programma infatti, o parte di esso, è possibile tramite la tastiera aggiungere, cancellare o modificare le istruzioni con le modalità che ora vedremo.
Per cancellare una linea di istruzione è sufficiente scrivere il numero di linea e premere 'RETURN'.
Per cancellare un carattere si deve portare il cursore (tramite i tasti

spostamento cursore descritti in seguito), sul carattere successivo e premere il tasto 'DEL'.
Per aggiungere un carattere si deve portare il cursore sul carattere seguente la posizione di inserzione e premere il tasto '(shift)INST' (cioè contemporaneamente i tasti 'SHIFT' e 'INST'); si noterà uno spostamento di tutti i caratteri a destra del cursore e l'introduzione di uno spazio bianco: si può quindi scrivere il carattere desiderato.

ESEMPIO

6. supponiamo di aver digitato la parola PINT anzichè PRINT.

Vogliamo inserire una R fra la P e la I. Si posizioni il cursore (tramite i tasti di spostamento cursore) sulla seconda I e si prema '(shift)INST': si ottiene

P INT

con il cursore sulla posizione vuota: ora premiamo il tasto 'R' e si avrà:

PRINT

Per modificare un carattere basta posizionare il cursore sopra di esso e scrivere il nuovo carattere.
Si faccia attenzione che corretta la visualizzazione della linea di programma, si deve premere il tasto RETURN affinchè le modifiche vengano riportate sul programma stesso.
Alcuni tasti sulla tastiera del PET, facilitano il posizionamento del cursore sul carattere desiderato: essi sono i tasti di spostamento cursore (a destra, a sinistra, in alto, in basso), e il tasto 'HOME' che posiziona il cursore in alto a sinistra.

editing sull'M20

Sul computer M20 il servizio di editing è un po' più disagevole e complesso, comunque sufficientemente ricco di possibilità di correzione.

Innanzi tutto si deve entrare in ambiente di EDIT per la linea di programma che si intende modificare, scrivendo:

EDIT /num. linea/

oppure:

EDIT .

Questo comando permette di entrare in ambiente di edit per la riga specificata da /num. linea/ (primo caso) o per la linea di programma attualmente considerata dalla macchina (secondo caso).

Entrata in edit. la macchina visualizza il numero di linea dell'istruzione specificata ed attende la selezione del tipo di correzione che si intende effettuare. Nella seguente tabella sono riportati i comandi di edit:

L (lista): visualizza lo stato corrente dell'istruzione e si riposiziona all'inizio della linea.

A (again): annulla tutte le eventuali modifiche già effettuate e si riposiziona in testa alla linea.

SPAZIATORE: visualizza il successivo carattere e sposta il cursore a destra di una posizione.

CTRL H: cancella l'ultimo carattere e sposta il cursore di una posizione a sinistra.

I (insert): entra nello stato di inserimento: tutti i caratteri successivamente digitati saranno inseriti nell'istruzione a partire dalla posizione corrente del cursore. Per uscire dallo stato di inserimento si deve digitare CTRL HOME.

X (extended line): provoca la visualizzazione della linea e l'attivazione dello stato di inserimento alla fine di essa.

CTRL HOME: esce dallo stato di inserimento, ma resta nel modo di edit, per ulteriori variazioni.

D (delete): cancella il carattere successivo alla posizione del cursore; i caratteri cancellati sono visualizzati fra una coppia di '/'.

n D (delete n characters): cancella n caratteri; n è un numero.

H (hack): cancella la parte rimanente della linea ed entra nello stato di inserimento.

S x (search for the 1st occurence): ricerca la prima occorrenza del carattere specificato a destra di S; il cursore si posiziona a sinistra di esso. Se non esiste tale carattere, il cursore si posiziona alla fine della linea.

n S x: ricerca l'ennesima occorrenza del carattere specificato.

D x (search and kill): cancella tutti i caratteri compresi fra la posizione del cursore e la prima occorrenza del carattere specificato.

n K x: cancella tutti i caratteri compresi fra la posizione del cursore e l'ennesima occorrenza del carattere specificato.

C x (change): rimpiazza il carattere successivo alla posizione del cursore, con il carattere specificato.

n C x1 x2 ... xn (change n characters): rimpiazza n caratteri.

RETURN: visualizza la linea con le modifiche apportate e ritorna al modo BASIC.

E (exit): ha lo stesso effetto di RETURN, ma la parte rimanente della linea non è visualizzata.

Q (quite): ritorna al modo BASIC annullando tutte le modifiche apportate alla linea di istruzione.

CAPITOLO 7

ISTRUZIONI BASIC

istruzione REM

REM (* /commento/ *)

Permette di inserire commenti all'interno di un programma.
L'istruzione REM non è eseguita, ma è stampata così come è introdotta, quando si stampa il listing del programma.
Non si possono inserire altre istruzioni sulla stessa riga dopo un commento, in quanto il simbolo separatore ':' è considerato commento.
Questa istruzione ha la stessa forma per entrambe le macchine.
Nell'M20 c'è inoltre la possibilità di definire un 'campo di commento' utilizzando la seguente forma:

' (* /commento/ *)

L'apostrofo ha lo stesso significato di REM, ma può essere scritto su una stessa linea di istruzione senza dover inserire il simbolo ':' di separazione.

istruzione LET

(* LET *) /variabile/=/espressione/

Assegna alla variabile a sinistra dell'uguale, il valore dell'espressione.
Notare che la parola LET è opzionale, e quindi il simbolo '=' è sufficiente per eseguire l'assegnazione. Questa istruzione è identica per entrambe le macchine.

ESEMPI:

LET D=12	è lo stesso che D=12
LET A=5*D−1	è lo stesso che A=5*D−1
LET B=3*(4−A)/D	è lo stesso che B=3*(4−A)/D

istruzione INPUT

COMMODORE: **INPUT (* "/stringa di suggerimento/"; *)**
/ lista di variabili/

con /lista di variabili/ descritta da:
/variabile/ (*, /variabile/ ,..., /variabile/ *)

Questa istruzione permette l'ingresso dei dati da tastiera durante l'esecuzione del programma.

Quando si esegue una istruzione di INPUT, il programma si interrompe e visualizza un punto interrogativo ad indicare che è in attesa di dati. Se è inclusa la stringa di suggerimento, essa è stampata prima del punto interrogativo.

La richiesta di dati va soddisfatta da tastiera.

I dati introdotti sono assegnati, nell'ordine, alla lista delle variabili: il numero dei dati introdotti (separati da virgola), deve essere lo stesso che il numero delle variabili nella lista. L'INPUT è limitato alla lunghezza di una linea-video.

Le variabili nella lista di INPUT possono essere sia numeriche che stringhe, naturalmente la serie di dati introdotti deve essere concorde con i tipi di dati specificati dalle variabili. Rispondendo all'INPUT con tipi sbagliati di valori, si ha il messaggio:

? REDO FROM START

Non viene fatto nessun assegnamento finchè non vengono forniti dati accettabili.

Rispondendo all'INPUT con troppi dati, si ha il messaggio:

? EXTRA IGNORED

ad indicare che i dati in più sono ignorati.

Con troppo pochi dati si ha:

??

ad indicare la necessità di soddisfare anche le rimanenti variabili.

M20: **INPUT (* ; *) (* "/stringa di suggerimento/" *)**
; ! , /lista di variabili/

Il significato dell'istruzione è simile al COMMODORE, con alcune differenze:
se viene specificato il carattere ';' prima della stringa di suggerimento, non viene cancellato tutto ciò che segue i dati digitati;
se al posto del ';' seguente la stringa di suggerimento si specifica il carattere ',', si sopprime il simbolo '?' sul video.
se i dati digitati alla richiesta di una INPUT non coincidono con il numero o il tipo delle variabili specificate, l'M20 emette il messaggio:

```
?Redo from start
```

ed è necessario reintrodurre tutti dati.

ESEMPI

7. riprendiamo l'esempio del programma PROG1.1, utilizzando l'istruzione INPUT per assegnare valori alle variabili:

```
10  REM PROG1.1TER:CALCOLO DI UNA ESPRESSIONE
20  INPUT A,B,C
30  R=B 2—4*A*C
40  PRINT"R="R
```

l'esecuzione del programma provoca la stampa di un punto interrogativo, al quale si deve rispondere con una terna di numeri separati dalla virgola (per esempio 3,4,5); fatto ciò l'esecuzione procede come nell'esempio precedente.

8. in questo esempio calcoliamo il quadrato del numero introdotto:

```
10  REM PROG1.2
20  INPUT X
30  PRINT X"AL QUADRATO É"X↑2
40  END
RUN
? 5 (il numero 5 è digitato e seguito da 'RETURN')
5 AL QUADRATO É 25
```

9. in questo esempio calcoliamo l'area del cerchio: come INPUT è richiesto il raggio

```
 5  REM PROG1.3:AREA DEL CERCHIO
10  PI=3.1415
20  INPUT "QUALE É IL RAGGIO";R
30  A=PI*R↑2
40  PRINT "L'AREA DEL CERCHIO É";A
RUN
QUALE É IL RAGGIO? 7.4 (digitare il num. 7.4)
L'AREA DEL CERCHIO É 172.0336
```

ESERCIZI

5. scrivere un programma che calcoli la media aritemetica di tre numeri introdotti da tastiera e visualizzi il risultato sul video.

6. scrivere un programma che risolva il seguente problema: sia data una vasca di 100 litri inizialmente riempita a metà, con due rubinetti di portate fisse introdotta da tastiera. Si calcoli il tempo in cui la vasca si riempie.

M20: **LINE INPUT.**

Questo comando è disponibile solo sul computer M20. É simile all'istruzione INPUT, ma permette di introdurre tutti i caratteri speciali (, ; . ecc.), fino al primo 'RETURN'.

istruzione GET

COMMODORE: **GET /variabile/**

Questa istruzione permette l'introduzione di un singolo carattere da tastiera.

Il carattere viene letto in un'area di memoria riservata alle funzioni di tastiera ('buffer' di tastiera); se non viene premuto alcun tasto alla variabile non viene assegnato alcun valore se è alfanumerica, o viene assegnato il numero '0' se è numerica; se viene premuto un tasto il carattere digitato viene assegnato alla variabile specificata.

La ricerca del carattere sul buffer della tastiera è velocissima e questo

non permette in generale la digitazione in tempo utile del tasto. Per questo motivo l'istruzione GET è solitamente usata nel ciclo seguente (che fa uso dell'istruzione IF...THEN, specificata in seguito):

```
100  A$=""
110  GET A$: IF A$="" THEN GOTO 110
```

Nell'istruzione 100 viene annullato l'eventuale contenuto di a*; nell'istruzione 110 viene esaminato il buffer della tastiera finchè non viene trovato un carattere, ovvero finchè non viene premuto un tasto.

Come esempio scriviamo un programma che usi l'istruzione GET per l'introduzione caratteri ed ogni carattere viene visualizzato:

```
10  A$=""
20  GET A$: IF A$="" THEN GOTO 20
30  PRINT A$;
40  RUN
```

M20: **INKEY$**

In questo computer il significato del comando COMMODORE-GET è assunto dalla parola INKEY. Il tipico formato di questa istruzione è:

/variabile/=INKEY$

Il funzionamento è del tutto simile al comando COMMODORE-GET.

istruzioni DATA, READ, RESTORE

DATA /lista di costanti/

con /lista di costanti/ descritta da /costante/ (* /costante/,...,/costante/ *)

READ /lista di variabili/

con /lista di variabili/ descritta da /variabile/ (*, /variabile/,...,/variabile/ *)

RESTORE

(Valido per entrambe le macchine).

Le istruzioni DATA memorizzano costanti (stringhe e numeri), che sono lette da istruzioni READ.

Una istruzione DATA non è eseguibile, e può contenere tante costanti

quante possono essere scritte su un'unica linea (separate da virgola). Si possono usare un qualsiasi numero di istruzioni DATA.

Le istruzioni READ accedono alle costanti di DATA, nell'ordine sia di numero di linea che di posizione.

La lista di costanti può contenere costanti di ogni tipo e formato, ma non sono permesse espressioni.

I tipi delle variabili nelle istruzioni READ devono corrispondere ai dati delle istruzioni DATA.

Il blocco dati definito dall'insieme delle istruzioni DATA può essere riletto, con istruzioni READ, purchè queste vengano precedute dall'istruzione RESTORE.

Questa istruzione ha come effetto l'azzeramento dell'apposito puntatore ai dati specificati dalle istruzioni DATA.

ESEMPI:

```
 80 READ A(1),A(2),A(3),A(4),A(5)
110 DATA 1.2,12.5,3,4.7,9.0
```

assegna le costanti di DATA all'array A.

```
40 READ A,B,C
50 READ P$,Q$
60 READ D1,D2,D3

.
210 DATA 3,–2,11,"AB"
220 DATA "CD",–8,0,10
```

Se vengono fatti dei tentativi di lettura tramite READ, senza corrispondenti istruzioni DATA, si visualizza il messaggio:

OUT OF DATA ERROR IN /numero di linea/

dove il /numero di linea/ è quello relativo all'istruzione READ che ha provocato l'errore.

ESERCIZIO

7. inserire in una istruzione DATA i nomi dei mesi e costruire un programma che prelevi questi nomi e li visualizzi sul video.

istruzione PRINT

PRINT /lista di espressioni/

con /lista di espressioni/ descritta da: /espr/ (* /sep/ /espr/.../sep/ /espr/ *)

dove /espr/ è una espressione numerica o alfanumerica e /sep/ è un separatore (, ; 'SPAZIATURA')

(Valido per entrambe le macchine).

Permette la visualizzazione di dati su video.

Se la lista di espressioni è omessa, viene stampata una riga vuota. Se invece è inclusa, i valori delle espresioni sono visualizzati sul video. Le espressioni possono essere numeriche o stringhe. Le stringhe devono essere incluse tra virgolette.

Le posizioni della stampa di ogni termine della lista, dipende dalla punteggiatura usata per la separazione. Il BASIC divide la linea di stampa in 10 zone: nella lista di espressioni la virgola causa la stampa del valore considerato nella zona successiva; il simbolo ';' causa la stampa del valore nella posizione immediatamente dopo il valore precedente.

Se l'ultimo termine della lista è seguito da ',' o da ';', la successiva istruzione PRINT continua sulla stessa linea. Se la stampa supera la lunghezza massima della linea di video, la stampa stessa continua sulla linea successiva.

Al posto della parola PRINT si può usare il simbolo '?'.

ESEMPI:

```
10  X=5
20  PRINT X—4,X+4,X*3
RUN
1    9    15
```

È possibile inserire nelle stringhe di stampa anche i movimenti del cursore e la cancellazione dello schermo.

ESEMPIO

10. come esempio disegnamo un quadrato di asterischi:

```
10  REM QUADRATO
20  KK$="***********"
30  PRINT "♥QQ]]]]" ;KK$
40  PRINT "]]]]*          *"
50  PRINT "]]]]*          *"
60  PRINT "]]]]*          *"
70  PRINT "]]]]*          *"
80  PRINT "]]]]" ;KK$
```

ESERCIZI

8. visualizzare al centro del video il proprio nome.
9. come l'esercizio 8, ma incorniciando il nome in un rettangolo.

M20: **WRITE (* /espressione/ *) (* ,/espressione/ *)**

Questa istruzione è disponibile solo sull'M20. Visualizza i dati risultati delle espressioni. Le espressioni possono essere di qualsiasi tipo (numeriche, logiche, relazionali, stringhe). Ogni espressione deve essere separata da virgola. I numeri vengono visualizzati con lo stesso formato dell'istruzione PRINT, ma non sono seguiti da spazi. Se è omessa la parte /espressione/ viene visualizzata una linea bianca.

ESEMPI:

```
A=52:B=21.5:C=13
WRITE A,B,C

52, 21.5, 13
```

istruzione GOTO

GOTO /numero di linea/

(Valido per entrambe le macchine).

Esegue un salto incondizionato ad uno specifico numero di linea. Se il numero di linea è una istruzione eseguibile il programma prosegue da quella linea, altrimenti dalla prima istruzione eseguibile seguente il numero di linea specificato.

Questa istruzione permette di saltare ovunque nel programma. Una buona

programmazione consiglia comunque di utilizzare il GOTO molto limitatamente e di saltare solo poche istruzioni. Un uso sconsiderato del GOTO infatti, porterebbe caoticità alla sequenzialità del programma, rendendolo di difficile lettura e correzione.

ESEMPIO

11. (NB: nel COMMODORE la variabile TI$ è riservata ed è incrementata automaticamente dall'orologio interno del computer).

```
10  REM OROLOGIO
20  INPUT"CHE ORA É :HHMMSS";TI$
30  PRINT "♥QQQQ]]]]]]" ;
40  PRINT "L' ORA É: "TI$;
50  PRINT "]▒▒▒▒▒▒▒▒▒▒▒▒▒]";
60  GOTO40
```

alla domanda: CHE ORA É:HHMMSS?, si deve rispondere digitando l'ora nella forma richiesta, cioè due cifre per l'ora, due per i minuti e due per i secondi; per esempio se sono le 9.15 e 4 secondi, si scriverà: 091504.

Per interrompere il programma si dovrà premere il tasto STOP.

istruzioni GOSUB, RETURN

GOSUB /numero di linea/

RETURN

(Valido per entrambe le macchine).

Le due istruzioni permettono rispettivamente il salto a, e il ritorno da subroutines (sottoprogrammi).

Il /numero di linea/ è il primo numero di linea di una subroutine. L'istruzione RETURN è l'ultima istruzione eseguita della subroutine. Provoca il salto di ritorno al programma chiamante, nel punto seguente alla chiamata.

Una subroutine può contenere più di una istruzione di RETURN, disposte in differenti punti.

Una subroutine può apparire ovunque nel programma, ma deve essere distinta da esso: per prevenire involontari ingressi ad una subroutine, questa

può essere preceduta da STOP, END, o da un GOTO che salta tutte le istruzioni.

Al massimo è permesso un annidamento di 23 subroutines.

ESEMPIO

12. il seguente programma è un esempio di utilizzo delle istruzioni GOTO e GOSUB.

```
 10  REM
 20  PRINT "♥"
 30  PRINT"DIAMETRO/LATO";
 40  INPUT D:R=D/2
 50  GOSUB100
 60  GOSUB200
 70  PRINT
 80  PRINT"NUOVO ";
 90  GOTO30
100  REM AREA CERCHIO
110  A=R*R*3.14
120  PRINT"L'AREA DEL CERCHIO É ";A
130  RETURN
200  REM AREA QUADRATO
210  B=D*D
220  PRINT"L'AREA DEL QUADRATO É ";B
230  RETURN
```

Per terminare questo programma e in generale per terminare un programma entrato in fase di INPUT dati, si deve digitare un 'RETURN' a vuoto (ovvero senza precedente digitazione di dato).

ESERCIZIO

10. si determini la circonferenza del cerchio e il perimetro del quadrato dell'esempio precedente, estendendolo anche al cerchio circoscritto al quadrato.

11. si costruisca un programma che chiama una subroutine che calcola la media e lo scarto dalla media di una sequenza di dati richiesti da tastiera nel main.

12. si costruisca un programma che chiama una subroutine che valuta la percentuale di riempimento di una vasca, essendo richiesti da programma principale le quantità progressive inserite.

istruzioni di SALTO CONDIZIONATO

ON /espressione/ GOTO /lista di numeri di linea/

ON /espressione/ GOSUB /lista di numeri di linea/

con /lista di numeri di linea/ descritta da: /numero di linea/ (* ,/numero di linea/,...*)

(Valido per entrambe le macchine).

É utilizzata per saltare ad uno dei diversi numeri di linea a seconda del risultato dell'espressione.

Per esempio se il valore dell'espressione è 3, allora si salta al terzo numero di linea specificato.

I numeri di linea nel secondo formato devono essere i primi numeri di linea di subroutines.

Se il valore dell'espressione è non intero, viene ignorata la parte decimale.

Se il valore è negativo, ci sarà l'errore:

ILLEGAL QUANTITY ERROR

Se il valore è 0, o maggiore dei termini della lista, il programma continua con l'istruzione seguente. Esempio:

```
100   ON L—1 GOTO 150,300,320,390
110
```

se L—1=2 salta all'istruzione 300
se L—1=5 salta all'istruzione 110

ESEMPIO

13. questo programma calcola l'area di un cerchio o l'area di un rettangolo o l'area di un trapezio a seconda del valore digitato da tastiera.

Inoltre a seconda del caso prescelto, chiede i particolari dati necessari a risolvere il problema.

```
 10  REM CERCHIO, RETTANGOLO, TRAPEZIO
 15  PRINT"♥"
 20  PRINT"QQ1=CERCHIO–2=RETTANGOLO–3=TRAPEZIOQ"
 25  INPUT"SCEGLIERE L'OPZIONE DESIDERATA" ;A
 30  PRINT"♥"
 40  ON A GOTO 100,200,300
 50  GOTO20
100  REM AREA CERCHIO
110  INPUT"RAGGIO" ;R
120  PRINT"AREA CERCHIO = "3.14*R*R
130  GOTO 20
200  REM AREA RETTANGOLO
210  INPUT"LATI DEL RETTANGOLO" ;L1,L2
220  PRINT"AREA RETTANGOLO = "L1*L2
230  GOTO20
300  REM AREA TRAPEZIO
310  INPUT"BASE MAGGIORE" ;B1
320  INPUT"BASE MINORE" ;B2
330  INPUT"ALTEZZA" ;H
340  PRINT"AREA TRAPEZIO"(B1+B2)*H/2
350  GOTO20
```

(NB. Se alla richiesta di dato nell'istruzione 20 si risponde digitando un numero negativo, il programma si interrompe).

ESERCIZIO

13. riscrivere il programma dell'esempio precedente utilizzando l'istruzione ON...GOSUB al posto dell'istruzione ON...GOTO.

istruzioni di TEST

COMMODORE: **IF /espressione/ THEN /istruzioni/ ! /numero di linea/**

con /istruzioni/ descritta da: /istruzione/ (* :/istruzione/ ... :/istruzione/ *)

IF /espressione/ GOTO /numero di linea/

Queste istruzioni sono usate per prendere decisioni riguardanti il flusso di programma, in base al risultato di una espressione.

Se il risultato è 'vero', allora la clausola THEN o GOTO è eseguita. La parte THEN può essere seguita o da un numero di linea o da una o più istruzioni (separate dal simbolo ':'). GOTO è sempre seguito da un numero di linea.

Se il risultato è 'falso', le clausole THEN e GOTO sono ignorate, e l'esecuzione continua dall'istruzione seguente l'IF.

L'istruzione IF...THEN può essere annidata, e il numero di annidamenti è limitato solo dalla lunghezza della linea.

ESEMPI:

```
50  IF A=B THEN IF B=C THEN PRINT "A=C"

40  INPUT M
50  IF (M>10)AND(M<20) THEN PRINT"NEL RANGE":GOTO 70
60  PRINT"FUORI DEL RANGE"
70  END
```

M20: **IF /espressione/ THEN /istruzioni/ ! /numero di linea/ (* ELSE /istruzioni/ ! /numero di linea/ *)**

con /istruzioni/ descritta da: /istruzione/ (* :/istruzione/ ... :/istruzione/ *)

IF /espressione/ GOTO /numero di linea/ (* ELSE/istruzioni/!/numero di linea/ *)

Nell'M20, il comando IF... ha la stessa operatività del COMMODORE, con l'aggiunta opzionale della parte ELSE, che viene eseguita solamente se l'espressione condizionale risulta falsa.

ESEMPIO

14. calcolo del fattoriale di un numero.

Il fattoriale è una funzione definita solo sui numeri interi maggiori o uguali a zero, e per un dato n (numero intero) è uguale al prodotto di tutti gli interi positivi minori o uguali a n; per esempio:

fatt(6)=6*5*4*3*2*1

inoltre si pone:

fatt(0)=1

Scriviamo il programma, organizzato a sottoprogramma, che calcola il fattoriale di un numero.

```
10   REM PROG1.6--FATTORIALE
20   INPUT"DAMMI UN NUMERO";N
30   IFN<0THENPRINT"ERRORE-IL NUMERO È NEGATI-
     VO":GOTO80
40   IFN<>INT(N)THENPRINT"ERRORE-IL NUMERO NON È
     INTERO": GOTO80
45   IFN>32THENPRINT"ERRORE-IL NUMERO È TROPPO
     GRANDE":GOTO80
60   IFP>1THENF=F*P:P=P-1 :GOTO60
55   IFN=0THENGOTO70
60   IFP>1THENF=F*P:P=P-1 :GOTO60
70   PRINT" IL FATTORIALE DI";N;" É";F
80   RETURN
```

La funzione INT usata nell'istruzione 40 restituisce il valore intero del numero passatole come parametro; per esempio INT(7.2) è uguale a 7.

Quindi N è diverso da INT(N) solamente quando N non è intero.
L'istruzione 45 è stata introdotta in quanto se si tentasse di calcolare il fattoriale di un numero maggiore di 32 si provocherebbe un errore di OVERFLOW.

ESEMPIO

15. in questo programma leggiamo un certo insieme di dati memorizzati con istruzioni DATA e li visualizziamo sul video insieme alla loro somma, alla loro media, al massimo e al minimo.
Il primo dato (istruzione 30) fornisce il numero di dati da leggere.
I dati da leggere sono introdotti con le istruzioni 40 e 50.
La visualizzazione dei dati avviene nella subroutine 1000-1020.

Il significato delle variabili usate è il seguente:

N = numero dei dati da leggere
T = somma dei dati
I = contatore
MI = minimo
MA = massimo
MD = media
A1 = variabile d'appoggio

```
  10  REM LETTURA DATI
  20  PRINT"♥"
  30  DATA 10
  40  DATA1.2,52.32,21.5,40,12.12
  50  DATA10.23,16,18.5,1.1,10
  60  READ N
  70  I=1
  80  READ A1
  90  GOSUB1000
 100  MI=A1:MA=A1:T=A1
 110  IF I =N THEN200
 120  I=I+1
 130  READ A1
 140  GOSUB1000
 150  IF A1<MI THENMI=A1
 160  IF A1>MA THENMA=A1
 170  T=1+A1
 180  GOTO110
 200  MD=T/N
 210  PRINT
 220  PRINT"IL TOTALE É"T
 230  PRINT"LA MEDIA É"MD
 240  PRINT"IL MASSIMO É"MA
 250  PRINT"IL MINIMO É"MI
 260  END
1000  REM VISUALIZZA DATO
1010  PRINT"DATO";I;":",A1
1020  RETURN
```

ESEMPIO

16. l'istruzione IF...THEN può essere usata anche per ordinare alfabeticamente dati alfanumerici.
Questo semplice programma richiede due dati alfanumerici da tastiera e li visualizza sul video in ordine alfabetico.

```
 10  INPUT A$
 20  INPUT B$
 30  IF A$>B$ THEN 100
 40  PRINT B$
 50  PRINT A$
 60  END
100  PRINT A$
110  PRINT B$
120  END
```

ESERCIZI

14. scrivere un programma che acquisisca dei dati numerici interi da tastiera. Se il dato in ingresso è pari viene sommato sulla variabile P; se è dispari sulla variabile D.
Quando viene digitato lo 0 vengono visualizzati i due parziali. Se il dato in ingresso è negativo o non intero viene ignorato.

15. scrivere un programma che risolva una equazione di secondo grado visualizzando l'eventuale forma complessa del risultato.

16. scrivere un programma che legga da tastiera 3 dati alfanumerici e li stampi ordinati alfabeticamente.
(Per confrontare alfabeticamente due dati alfanumerici si possono usare gli stessi operatori relazionali usati per i confronti fra numeri. Per esempio se si desidera sapere quale fra le due variabili A$ e B$ precede l'altra secondo l'ordinamento alfabetico, è sufficiente testare l'espressione:

A$ > B$

se il risultato è vero, A$ contiene una stringa alfabeticamente precedente a quella contenuta da B$. Se l'espressione è falsa A$ segue od è uguale a B$).

17. scrivere un programma che calcoli la radice quadrata (approssimata a meno dell'1%) di un numero senza l'utilizzo della funzione SQR. (La si calcoli per approssimazioni successive).

Istruzione Iterativa

FOR...STEP...NEXT

FOR /variabile/=/espr.x/ TO /espr.y/ (* STEP /espr.z/ *)

.

NEXT (* /variabile/ *) (* ,/variabile/ *)

Con questa istruzione, valida per entrambe le macchine, è possibile eseguire in un ciclo un gruppo di istruzioni. /variabile/ è usata come contatore. L'espressione numerica /espr.x/ è il valore iniziale del contatore. L'espressione numerica /espr.y/ è il valore finale.

Le linee di programma seguenti il FOR sono eseguite finchè non è trovata una istruzione NEXT. A questo punto il contatore è incrementato del valore dell'espressione numerica /espr.z/, specificata da STEP. Se la parte STEP è assente, il contatore è incrementato automaticamente di 1. Il valore /espr.z/ può anche essere un numero non intero. Ci sarà poi un confronto per controllare se il valore attuale del contatore è diventato più grande del valore finale /espr.y/. Se non lo è, si ritorna indietro ad eseguire tutte le istruzioni comprese tra il FOR e il NEXT. Se il confronto rivela che il contatore è più grande di /espr.y/, l'esecuzione prosegue con l'istruzione che segue il NEXT.

Se STEP è negativo, il valore finale /espr.y/ deve essere minore di quello iniziale, e il ciclo FOR...NEXT è ripetuto decrementando il contatore.

ESEMPIO

17. questo programma disegna sul video una funzione incorniciata in un rettangolo di asterischi. Si usa la funzione SIN che restituisce il seno del numero passatole come parametro.
Il programma richiede il numero di colonne del video (digitare 40 oppure 80 a seconda del computer a disposizione) necessario per dimensionare il rettangolo, quindi richiede gli estremi dell'intervallo desiderato e il passo di incremento. A questo punto disegna il rettangolo (subroutine 2000-2070) e nel suo interno la funzione calcolata nell'intervallo e con il passo introdotti.

Si provi ad eseguire il programma con:

primo estremo	0
secondo estremo	3.14
passo	0.2

```
  10  REM FUNZIONE
  15  INPUT"COLONNE VIDEO";L2
  16  L1=20
  20  INPUT"PRIMO ESTREMO";A
  30  INPUT"SECONDO ESTREMO";B
  40  INPUT"PASSO";C
  50  GOSUB2000:REM DISEGNA QUADRO
  60  MN=SIN(A):MX=SIN(A)
  70  FOR X=A TO B STEP C
  80  Y=SIN(X)
  90  IF Y>MX THEN MX=Y
 100  IF Y<MN THEN MN=Y
 110  NEXT X
 120  PS=(MX-MN)/L1
 130  IX=0
 140  FOR X=A TO B STEP C
 150  IX=IX+1
 160  IY=L1-INT((SIN(X)-MN)/PS)
 170  GOSUB1000
 180  NEXT X
 190  PRINT"SQQQQQQQQQQQQQQQQQQQQ"
 200  END
1000  PRINT"S ";
1010  FOR I=1 TO IY: PRINT"Q" ; :NEXT I
1020  FOR I=1 TO IX: PRINT"]" ; :NEXT I
1030  PRINT "*"
1040  RETURN
2000  PRINT"♥*" ;
2010  FORI=1 TOL1+1:PRINT"Q]*" ; :NEXT
2020  PRINT"*" ;
2030  FORI=1TOL2-3:PRINT"*" ; :NEXT
2040  PRINT"S*" ;
2050  FORI=1TOL2-3:PRINT"*" ; :NEXT
2060  FORI=1TOL1+1:PRINT"Q]*" ; :NEXT
2070  RETURN
```

18. scrivere un programma che calcoli il fattoriale di un numero utilizzando l'istruzione FOR...NEXT.

19. scrivere un programma che calcoli la somma dei quadrati dei primi N numeri interi, con N digitato da tastiera.

20. scrivere un programma che approssimi il valore di PI-GRECO ricordando che la circonferenza è l'elemento separatore fra i poligoni inscritti e circoscritti.
(Traccia: raggio unitario. Il quadrato inscritto ha perimetro 5.65..., quello circoscritto 8. La circonferenza 2*PI-GRECO).

21. scrivere un programma che calcoli lo zero di una funzione usando il metodo di bisezione.
(Traccia: si prendano due punti di partenza, uno a destra e uno a sinistra del punto di annullamento; ci si avvicina a passi discreti).

22. scrivere un programma che, dati in ingresso il capitale iniziale e l'interesse, calcoli il capitale totale di ogni anno per i primi 40 anni e visualizzi questi risultati in un'unica pagina di video.

COMMODORE: **CLR**

M20: **CLEAR**

Mette tutte le variabili numeriche a 0, annulla tutte le stringhe, riinizializza la "fine della memoria" e lo spazio della pila, infine libera lo spazio assegnato alle variabili di array.
Questo comando può essere eseguito anche da programma, ma si deve fare molta attenzione affinchè non vengano provocati danni che impediscono la continuazione (in particolare per le subroutine, in quanto usano la pila).
Questo comando è utilizzato per la pulizia della memoria dati da programmi, utile con le tecniche di OVERLAY (aggancio automatico di programmi — vedi oltre).

M20: **ERASE**

ERASE /array/ (* ,/array/ *)

Questo comando permette la cancellazione di una o più matrici per eventuali ridimensionamenti o recupero spazio di memoria.
É presente solo nel computer M20.

COMMODORE: **SYS**

SYS /variabile/ (* (/lista di argomenti/) *)

con /lista di argomenti/ descritta da: /costante/!/variabile/ (* ,/costante/!/variabile/...)

É utilizzata per chiamare una subroutine scritta in linguaggio assembler. (Vedi anche la funzione USR).
La /variabile/ deve contenere l'indirizzo del punto di partenza in memoria della subroutine: non può essere una variabile di array.
La /lista di argomenti/, contiene la lista degli argomenti che sono passati alla subroutine.

ESEMPIO:

```
110  MIASUB=30200
120  SYS MIASUB (AB)
```

passa il controllo dell'esecuzione alla subroutine ASSEMBLER che inizia nella locazione di memoria 30200, con il parametro AB (che è una variabile precedentemente inizializzata).

M20: **CALL**

CALL /nome/ (* (/lista argomenti/) *)

Esegue la subroutine chiamata /nome/ scritta in linguaggio assembler o appartenente al sistema operativo PCOS. Gli argomenti nella lista devono essere separati da virgola. Se un argomento è di uscita (cioè conterrà un valore dopo l'esecuzione della subroutine) dovrà essere preceduto dal simbolo '@'.

ESEMPIO:

```
A$="LIST"
CALL "PK" (&41,A$)
```

L'esecuzione di queste due istruzioni assegna al tasto '.' la parola 'list'.
Il simbolo & specifica una costante esadecimale.

WAIT

COMMODORE: **WAIT /indirizzo/ ,I (* ,J *)**

L'istruzione WAIT sospende l'esecuzione del programma finchè uno specificato indirizzo di macchina presenta una specificata configurazione di bit. Il dato letto all'indirizzo, è messo in OR-esclusivo con l'espressione intera J, ed è messo in AND con I. Se il risultato è 0, si ritorna a leggere un altro dato all'indirizzo specificato; se il risultato è non zero, l'esecuzione continua con la successiva istruzione.

L'OR - esclusivo è definito:

X	Y	X OR-EX Y
0	0	0
1	0	1
0	1	1
1	1	0

ATTENZIONE: è possibile entrare in un ciclo senza termine con questa istruzione nel qual caso si deve riinizializzare manualmente la macchina.
L'istruzione WAIT è prevalentemente utilizzata per il colloquio con unità periferiche non standard. La macchina resta in attesa che l'unità periferica esterna gli passi dei dati modificando la locazione designata.

ESEMPIO

18. l'istruzione seguente:

```
100 WAIT 59411,8,8
```

sospende l'esecuzione del programma finchè non è premuto il tasto PLAY dell'unità a nastro (premendo quel tasto viene messo un particolare dato nella locazione 59411).

CAPITOLO 8

LA PREPARAZIONE DI UN DIAGRAMMA DI FLUSSO (FLOW-CHART)

Prima di scrivere un programma, è buona norma tracciare uno schema che rappresenti graficamente la sua logica.

Lo schema che discuteremo è il cosiddetto flow-chart (o schema di flusso, o diagramma di flusso).

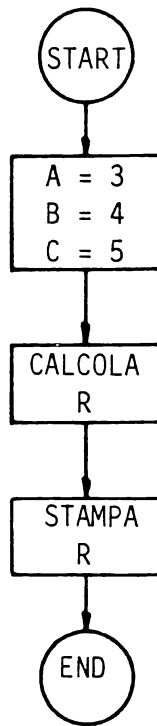
Il flow-chart è uno schema formato da figure geometriche di diverso formato, unite da archi orientati. A seconda della loro forma, le figure assumono particolari significati:

le figure circolari indicano inizio o termine di programmi

le figure rettangolari indicano blocchi operativi

le figure romboidali indicano blocchi decisionali.

Partendo dal blocco di inizio (start), seguendo i segmenti orientati, si controlla il cammino compiuto dal calcolatore durante l'esecuzione del programma.

Tracciamo il flow-chart (in cui non compaiono blocchi decisionali), del programma PROG1.1.

Tracciamo ora il flow-chart di un programma che calcola le radici di una equazione di secondo grado (soluzione dell'esercizio 15.parte seconda) nella forma:

$AX^2+BX+C=0$

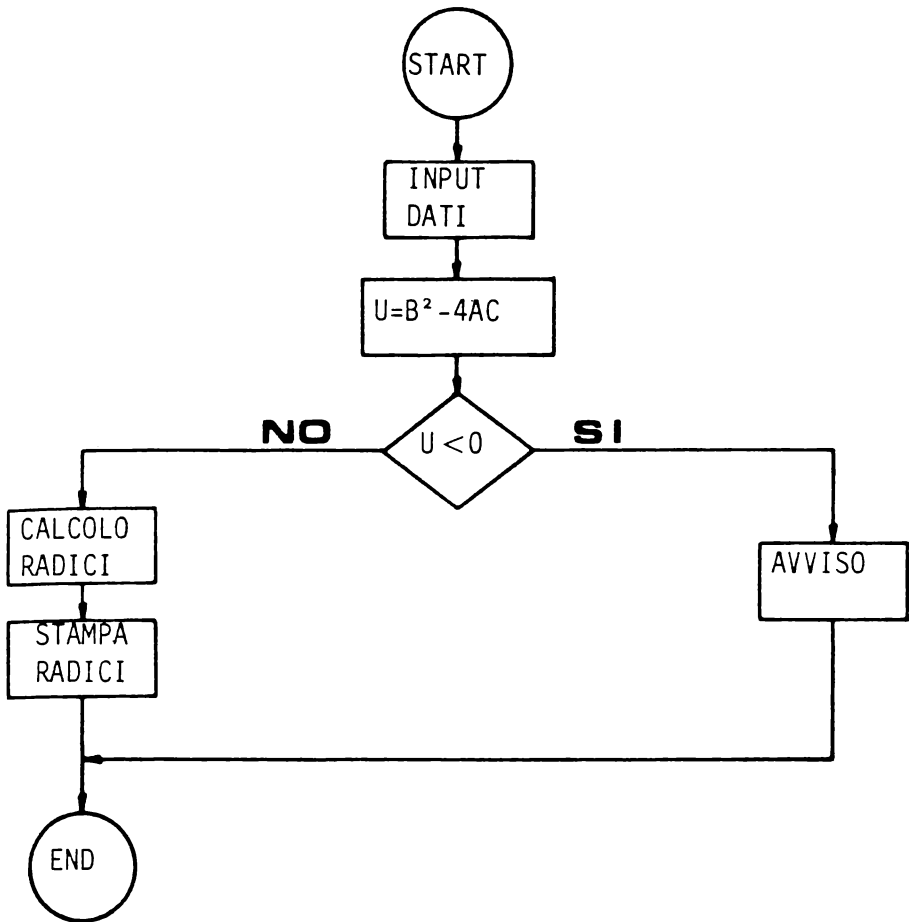
Le radici sono calcolate semplicemente dalle espressioni:

$X1=(-B-\sqrt{B^2-4AC})/(2A)$

$X2=(-B+\sqrt{B^2-4AC})/(2A)$

ma bisogna controllare che il termine B^2-4AC sia non negativo, per non dare alla funzione SQR (che calcola la radice quadrata), un valore negativo, con conseguente visualizzazione di:

? ILLEGAL QUANTITY ERROR

Ecco il diagramma:

Da un flow-chart è facile ricavare il programma corrispondente:

```
10  REM EQUAZIONE SECONDO GRADO
20  INPUTA,B,C
30  U=B*B–4*A*C
40  IFU<0THENPRINT"RADICI NON REALI":GOTO70
50  X1=(–B–SQR(U))/(2*A)
60  X2=(–B+SQR(U))/(2*A)
65  PRINTX1,X2
70  END
```

Introduciamo il programma nella memoria del computer (ricordando di digitare prima il comando NEW).

Ora possiamo risolvere automaticamente le equazioni di secondo grado. Per esempio consideriamo l'equazione:

$X^2+5X+6=0$

Per trovare le sue radici, scriviamo RUN e all'apparire del punto interrogativo (che indica richiesta dati) scriviamo i numeri 1,5,6 (separati dalla virgola): compariranno i valori

−3 −2

cioè le radici della precedente equazione.

Ora riscriviamo RUN e digitiamo i numeri 4,3,5: il risultato sarà il messaggio

RADICI NON REALI

il che significa che l'equazione $4X^2+3X+5=0$ non ha radici reali.

Un'ultima particolarità: con alcune terne di dati in input, il risultato finale non è esattamente la coppia di valori teorici aspettati, ma approssimazioni di questi. La spiegazione a questo fatto è da ricercare nelle approssimazioni interne di calcolo e di rappresentazione dei numeri floating point.

ESERCIZI

23. disegnare il flow-chart dei programmi finora svolti seguendone il flusso logico.

CAPITOLO 9

FUNZIONI NUMERICHE

COMMODORE–M20: **ABS(X)**

calcola il valore assoluto di X, cioè:

  X se X>=0
–X se X<0

COMMODORE–M20: **INT(X)**

calcola l'intero più grande minore o uguale a X.

ESEMPIO:

INT(12.8)=12
INT(– 4.3)=–5

COMMODORE–M20: **SGN(X)**

valuta il segno di X:

se X>0 SGN(X)=1
se X=0 SGN(X)=0
se X<0 SGN(X)=–1

ESEMPIO:

```
ON SGN(X)+2 GOTO 100,200,300
```

salta a 100 se X<0
salta a 200 se X=0
salta a 300 se X>0

COMMODORE–M20: **SQR(X)**

calcola la radice quadrata di X.
X deve essere maggiore o uguale a 0

COMMODORE–M20: **FRE(X)**

L'argomento della FRE è solo di comodo. Essa fornisce il numero di bytes liberi nella memoria (non usati dal BASIC). Dopo aver introdotto un programma in memoria, se desideriamo conoscere la memoria ancora disponibile, si deve digitare:

PRINT FRE(0)

COMMODORE **RND(X)**

Questa funzione è utilizzata per generare numeri pseudo-causali tra 0 e 1 (con 0 compreso e 1 escluso, cioè il numero 0 è generabile mentre il numero 1 non potrà mai essere generato).

Se l'argomento fornito alla funzione é positivo, il valore restituito é indipendente da esso.

Se invece é negativo, il valore restituito dipende dall'argomento fornito.

Se X=0 si genera un numero da un clock interno.

L'utilità di questa funzione va ricercata in applicazioni di tipo statistico e in giochi col computer.

M20: **RND**

Nell'M20 viene generata sempre la stessa sequenza random; se la si desidera cambiare si deve scrivere (prima di RND), il comando RANDOMIZE, e digitare un numero compreso nell'intervallo –32768, +32767. Questo comando inizializza il generatore di numeri ad una nuova sequenza.

Anche nell'M20 i numeri generati ad ogni chiamata di RND sono compresi nell'intervallo 0, +1.

ESEMPIO

19. il programma che segue genera 'n' numeri interi pseudo-casuali compresi fra '0' ed 'm'.

```
  10  REM GENERATORE DI INTERI
  20  PRINT"♥QQQQ"
  30  INPUT"QUANTI NUMERI" ; N
  40  INPUT"MINORI DI" ;M
  45  PRINT"♥"
  50  FOR I=1 TO N
  60  GOSUB1000
  70  PRINTX,
  80  NEXT
  90  END
1000  REM GENERATORE
1010  X=INT (RND(1)*M)
1020  RETURN
```

ESEMPIO

20. questo programma utilizza il generatore del precedente esempio per simulare il lancio di un dado.

```
  10  REM DADO
  20  PRINT"♥"
  30  X=INT (RND (1 )*6+1 )
  40  ON X GOSUB 1000,2000,3000,4000,5000,6000
  50  PRINT"QQQ"
  60  INPUT"UN ALTRO LANCIO (S/N)" ; A$
  70  IF A$="S" OR A$="SI" THEN20
  80  END
1000  PRINT"]]]]]]]]]]R            "
1010  PRINT"]]]]]]]]]]R            "
1020  PRINT"]]]]]]]]]]R     ■      R  "
1030  PRINT"]]]]]]]]]]R            "
1040  PRINT"]]]]]]]]]]R            "
1100  RETURN
2000  PRINT"]]]]]]]]]]R            "
```

```
2010  PRINT"]]]]]]]]]]R  ■   R        "
2020  PRINT"]]]]]]]]]]R            "
2030  PRINT"]]]]]]]]]]R      ■   R   "
2040  PRINT"]]]]]]]]]]R            "
2100  RETURN
3000  PRINT"]]]]]]]]]]R            "
3010  PRINT"]]]]]]]]]]R ■   R       "
3020  PRINT"]]]]]]]]]]R    ■     R  "
3030  PRINT"]]]]]]]]]]R      ■   R  "
3040  PRINT"]]]]]]]]]]R            "
3100  RETURN
4000  PRINT"]]]]]]]]]]R            "
4010  PRINT"]]]]]]]]]]R  ■   R  ■   R   "
4020  PRINT"]]]]]]]]]]R            "
4030  PRINT"]]]]]]]]]]R  ■   R  ■   R   "
4040  PRINT"]]]]]]]]]]R            "
4100  RETURN
5000  PRINT"]]]]]]]]]]R            "
5010  PRINT"]]]]]]]]]]R  ■   R  ■   R   "
5020  PRINT"]]]]]]]]]]R    ■  R     "
5030  PRINT"]]]]]]]]]]R  ■   R  ■   R   "
5040  PRINT"]]]]]]]]]]R            "
5100  RETURN
6000  PRINT"]]]]]]]]]]R            "
6010  PRINT"]]]]]]]]]]R■   R    ■   R   "
6020  PRINT"]]]]]]]]]]R■   R    ■   R   "
6030  PRINT"]]]]]]]]]]R■   R    ■   R   "
6040  PRINT"]]]]]]]]]]R            "
6100  RETURN
```

ESERCIZI

24. Costruire un generatore di numeri interi compresi tra −10 e +10.
Costruire un generatore di numeri interi positivi dispari minori di 100.

25. Generare 100 numeri casuali tra 0 e 1; calcolare quindi la media e la varianza.
(La varianza è definita con la relazione:

$$V = \sum_i \frac{(Xi - MD)}{N}$$

Ove Xi rappresenta il valore dell'i-esimo numero, MD rappresenta il valore medio dei numeri generati ed N la quantità dei numeri generati (in questo caso 100).

COMMODORE–M20: **COS(X)**

calcola il coseno di x in radianti

COMMODORE–M20: **EXP(X)**

calcola e (numero di Nepero: 2.71828183...) elevato ad X: X deve essere <=88.02969191.

COMMODORE–M20: **LOG(X)**

calcola il logaritmo naturale di X: X deve essere maggiore di 0.

COMMODORE–M20: **SIN(X)**

calcola il seno di X in radianti

COMMODORE–M20: **TAN(X)**

calcola la tangente di X in radianti.

COMMODORE–M20: **ATN(X)**

calcola l'arcotangente di X in radianti; il risultato é nel range –pi/2, pi/2

ESEMPIO

21. questo programma disegna sul video una circonferenza.

```
 0  REM DISEGNO
 1  PRINT"♥"
10  A$="S"
20  FORI=0TO6.28STEP.2
30  PRINTA$
40  G=20+INT (COS (I )*10 )
50  H=12+INT ( SIN ( I )*10 )
60  FORJ=0TOG: PRINT"]" ; : NEXTJ
70  FORK=0TOH: PRINT"Q" ; : NEXTK
```

```
 80  PRINT"*"
 90  NEXTI
100  END
```

ESEMPIO

22. questo programma visualizza sullo schermo la tabella relativa ad alcune funzioni numeriche. Attribuiamo all'argomento delle funzioni valori compresi fra 0 e PI - GRECO.

```
10  REM TABELLA DI FUNZIONI
15  PRINT"♥"
20  PRINT"   X","   SIN(X)","      TAN(X)","       COS(X)"
30  PRINT
40  FOR X=0 TO 3.14 STEP 0. 1
50  PRINTX, SIN(X), TAN(X), COS(X)
60  NEXT X
```

ESERCIZI

26. Scrivere un programma che visualizzi il grafico della funzione tangente e della parabola.

27. Scrivere un programma che tabuli la funzione Y=LOG(X*X), per valori di X compresi fra −1 e +1 e passo 0.1. Si faccia attenzione ad e-vitare di calcolare il logaritmo con argomento nullo.

CAPITOLO 10

FUNZIONI PER IL TRATTAMENTO DELLE STRINGHE

COMMODORE–M20: **CHR$ (I)**

restituisce il carattere il cui codice ASCII é I. Questa funzione è comunemente usata per specificare caratteri che non possono essere rappresentati in stringa. Questi caratteri sono per esempio i doppi apici (nel COMMODORE), il carattere 'RETURN', ecc.

ESEMPIO:

```
PRINT CHR$ (34) "PATATRACH" CHR$ (34)
```

visualizza "PATATRACH" tra doppi apici.
Si veda in appendice la tabella dei codici ASCII.

COMMODORE–M20: **ASC (X$)**

restituisce il valore numerico che é il codice ASCII del primo carattere della stringa X$. Se X$ è la stringa nulla viene visualizzato:

"ILLEGAL QUANTITY"

ESEMPIO:

```
10  X$="TESTO"
20  PRINT ASC (X$)
30  PRINT CHR$ (84)
40  END
```

```
RUN
84
T
READY
```

COMMODORE–M20: **STR$ (I)**

trasforma l'argomento numerico X, in una stringa;

ESEMPIO:

```
10  A=11
20  A$=STR$ (A)
```

COMMODORE–M20: **VAL (X$)**

trasforma la stringa X$ nel suo valore numerico: se il primo carattere non è +, –, $ o un numero, allora VAL (X$)=0.

ESEMPIO

```
se X$="+12. 3"    VAL (X$)=12. 3
se X$="77AC9"     VAL (X$)=77
se X$="G7783"     VAL (X$)=0
```

Funzioni per la manipolazione delle stringhe

COMMODORE–M20: **LEN (X$)**

restituisce il numero dei caratteri componenti la stringa X$ (ovvero la sua lunghezza);

ESEMPIO:

```
LEN ("ABCD")=4
LEN ("A 25 GB")=7
```

COMMODORE–M20: **LEFT$ (X$,n)**

restituisce gli n caratteri più a sinistra di X$. n deve essere nel range 0, 255.

Se n é più grande della lunghezza della stringa, questa funzione associa l'intera stringa. Se n=0 é associata la stringa nulla.

ESEMPIO

23. esempio d'uso di LEFT$

```
10  REM PROVA LEFT$
20  A$="ABCDEFGHILMNOPQRSTUVZ"
25  PRINT "♥"
30  FOR I=1 TO 21
40  B$=LEFT$ (A$ , I )
50  PRINTB$
60  NEXT I
```

Il risultato che compare sul video é

```
A
AB
ABC
ABCD
ABCDE
ABCDEF
ABCDEFG
ABCDEFGH
ABCDEFGHI
ABCDEFGHIL
ABCDEFGHILM
ABCDEFGHILMN
ABCDEFGHILMNO
ABCDEFGHILMNOP
ABCDEFGHILMNOPQ
ABCDEFGHILMNOPQR
ABCDEFGHILMNOPQRS
ABCDEFGHILMNOPQRST
ABCDEFGHILMNOPQRSTU
ABCDEFGHILMNOPQRSTUV
ABCDEFGHILMNOPQRSTUVZ
```

COMMODORE–M20: **MID$ (X$,n, (*,m *))**

restituisce una stringa di m caratteri, che inizia dall'n-esimo carattere. n ed m devono essere nel range 0, 255.

Se m é omesso, o se m é più grande del numero di caratteri a destra dell'n -esimo (contando anche l'n-esimo), sono presi tutti i caratteri a destra dell'n-esimo. Se n>LEN (X$), la funzione associa la stringa vuota.

ESEMPIO

24. esempio d'uso di MID$

```
10  REM PROVA MID$
20  A$="ABCDEFGHILMNOPQRSTUVZ"
25  PRINT "♥"
30  FOR I=1 TO 21
40  B$=MID$ (A$, I )
50  PRINTB$
60  NEXT I
```

Il risultato che compare sul video é

```
ABCDEFGHILMNOPQRSTUVZ
BCDEFGHILMNOPQRSTUVZ
CDEFGHILMNOPQRSTUVZ
DEFGHILMNOPQRSTUVZ
EFGHILMNOPQRSTUVZ
FGHILMNOPQRSTUVZ
GHILMNOPQRSTUVZ
HILMNOPQRSTUVZ
ILMNOPQRSTUVZ
LMNOPQRSTUVZ
MNOPQRSTUVZ
NOPQRSTUVZ
OPQRSTUVZ
PQRSTUVZ
QRSTUVZ
RSTUVZ
STUVZ
TUVZ
UVZ
VZ
Z
```

25. esempio d'uso di MID$

```
10  REM PROVA MID$
20  A$="ABCDEFGHILMNOPQRSTUVZ"
25  PRINT"♥"
30  FOR I=1 TO 18
40  B$=MID$ (A$, I , 4 )
50  PRINTB$
60  NEXT I
```

Il risultato che compare sul video é

```
ABCD
BCDE
CDEF
DEFG
EFGH
FGHI
GHIL
HILM
ILMN
LMNO
MNOP
NOPQ
OPQR
PQRS
QRST
RSTU
STUV
TUVZ
```

COMMODORE–M20: **RIGHT$ (X$, n)**

Restituisce gli n caratteri più a destra di X$. Se n=0 allora la funzione ritorna la stringa vuota (di lunghezza zero).

ESEMPIO

26. esempio d'uso di RIGHT$

```
10  REM PROVA RIGHT$
20  A$="ABCDEFGHILMNOPQRSTUVZ"
25  PRINT "♥"
30  FOR I=1 TO 21
40  B$=RIGHT$ (A$ , I)
50  PRINTB$
60  NEXT I
```

Il risultato che comparirà sul video è il seguente

```
Z
VZ
UVZ
TUVZ
STUVZ
RSTUVZ
QRSTUVZ
PQRSTUVZ
OPQRSTUVZ
NOPQRSTUVZ
MNOPQRSTUVZ
LMNOPQRSTUVZ
ILMNOPQRSTUVZ
HILMNOPQRSTUVZ
GHILMNOPQRSTUVZ
FGHILMNOPQRSTUVZ
EFGHILMNOPQRSTUVZ
DEFGHILMNOPQRSTUVZ
CDEFGHILMNOPQRSTUVZ
BCDEFGHILMNOPQRSTUVZ
ABCDEFGHILMNOPQRSTUVZ
```

ESEMPIO

27. questo programma genera parole casuali di 6 lettere

```
10  REM GENERATORE DI PAROLE
15  PRINT"♥"
20  C$="BCDFGLMNPRSTVZ"
30  V$="AEIOU"
```

```
  35  FOR J=1 TO100
  40  U$=" "
  50  FORI=1 TO3
  60  GOSUB1000
  70  A=INT (X*14) + 1
  80  U$=U$+MID$ (C$ , A , 1)
  90  GOSUB1000
 100  A=INT (X*5) + 1
 110  U$=U$+MID$ (V$ , A , 1)
 120  NEXT I
 130  PRINTU$,
 140  NEXT J
 150  END
1000  X=RND (0) : RETURN
```

Operazioni su stringhe

Le stringhe possono essere concatenate usando il simbolo '+' ; per esempio:

```
10  A$="CO.R. " :B$="EL. " :C$="IULI" :D$="FR"
20  E$=A$+B$+"      "+D$+C$
30  PRINT E$
RUN
CO.R.EL.     FRIULI
```

Le stringhe possono essere confrontate usando gli stessi operatori relazionali usati per i numeri:

=, <, <>, <=, >=

i confronti sono effettuati prendendo ordinatamente un carattere alla volta da entrambe le stringhe, e confrontando i rispettivi codici ASCII. Se i codici corrispondono per entrambe le stringhe, allora esse sono uguali.

Se durante i confronti dei caratteri, una stringa termina, essa viene considerata minore. I blanks sono significativi.

ESEMPIO

```
"AABB" < "AB"
"UDINE" = "UDINE"
"UDINE" < "UDINE "
"AA A" < "AAA"
```

ESEMPIO

28. questo programma inverte una stringa data in ingresso

```
10  REM INVERSIONE STRINGA
15   PRINT"♥"
20  INPUT "STRINGA" ; A$
30  L=LEN (A$) : B$=" "
40  FOR I=L TO 1 STEP -1
50  B$=B$+MID$ (A$ , I , 1)
60  NEXT I
70  PRINT "QQQ"
80  PRINTA$ CHR$ (13) B$
```

(Si noti che l'istruzione 80 é equilavente alla coppia di istruzioni:

```
70  PRINT A$
80  PRINT B$
```

Infatti CHR$ (13) è il carattere 'RETURN' ovvero il carattere che posiziona a capo il cursore).

ESERCIZI

28. si scriva un programma che visualizzi tutte le permutazioni possibili delle lettere che formano una data parola introdotta da tastiera.

29. scrivere un programma che data una stringa in ingresso visualizzi un suo anagramma generato casualmente.

30. scrivere un programma che data una stringa in ingresso visualizzi tutte le sue sottostringhe di 1, 2 e 3 caratteri.

CAPITOLO 11

GESTIONE MATRICI

COMMODORE–M20:

DIM /variabile indiciata/ (* ,/variabile indiciata/...*)

con /variabile indiciata/ descritta da:
/variabile/ (/numero/ (* ,/numero/...*))

Questa istruzione è utilizzata per assegnare memoria sufficiente alle variabili di array (matrici).

Se una variabile è usata senza una istruzione DIM, viene automaticamente assegnata una dimensione massima di 10.

Se un indice di array supera il massimo dichiarato, si ha un messaggio d'errore:

BAD SUBSCRIPT ERROR IN /numero di linea/

Il minimo valore degli indici è sempre 0.

L'istruzione DIM mette tutti gli elementi dell'array a 0. Le matrici possono avere fino a 255 dimensioni, ma l'effettiva ampiezza dipende dalla memoria disponibile.

ESEMPIO:

```
DIM A(20),R3(5,5),DD$(2,2,4)
```

definisce le matrici A, R3, DD$

Nell'M20 vi è la possibilità di cancellare una matrice (e quindi ridimensionare se desiderato), tramite il comando:

ERASE /lista di array/

dove /lista di array/ è una serie di nomi di array, separati da virgola.
Inoltre nell'M20, si può anche definire il minimo valore dell'indice di array, scelto tra 0 o 1, tramite il comando:

OPTION BASE 0!1

Selezionando 1, il minimo valore degli indici di array diventa il numero 1.

ESEMPIO

29. questo esempio richiede 16 numeri da tastiera e li memorizza in un vettore.

Quindi calcola la loro media e li visualizza insieme al loro scarto.

```
10   REM VETTORE
20   DIM A(16):T=0
30   FOR I=1 TO 16
35   PRINT"♥NUMERO"I;
40   INPUTA(I):T=T+A(I)
50   NEXT I
60   GOSUB1000:REM CALCOLO MEDIA
70   GOSUB2000:REM VISUALIZZAZIONE
80   END
1000 REM CALCOLO MEDIA
1010 MD=T/16
1020 RETURN
2000 REM VISUALIZZAZIONE
2005 PRINT"VALORE","SCARTO"
2010 FOR I=1 TO 16
2020 PRINTA(I),A(I)-MD
2030 NEXT I
2040 RETURN
```

ESEMPIO

30. questo programma richiede in ingresso 'n' numeri e li riordina in modo crescente in un vettore; quindi li stampa.

```
  10  REM SORT
  20  INPUT"♥QUANTI NUMERI";N
  30  DIM M(N)
  35  PRINT:PRINT
  40  FOR I=1 TO N
  45  PRINT"♥NUMERO"I;
  50  INPUTM(I)
  60  NEXT I
  70  GOSUB1000:REM ORDINAMENTO
  80  GOSUB2000:REM VISUALIZZAZIONE
  90  END
1000  REM ORDINAMENTO
1020  FOR I=1 TO N
1030  FOR J=I TO N
1040  IF M(J)<M(I) THEN GOSUB3000
1050  NEXT J,I
1060  RETURN
2000  REM VISUALIZZAZIONE
2010  FOR I=1 TO N
2020  PRINTM(I)
2030  NEXT I
2040  RETURN
3000  REM INVERSIONE POSIZIONI
3010  A=M(I):M(I)=M(J):M(J)=A
3020  RETURN
```

ESERCIZI

31. scrivere un programma che ricerca il massimo e il minimo in un vettore di numeri.

32. scrivere un programma che ordini i numeri memorizzati in un vettore in modo decrescente.

33. come il precedente, ma considerando i valori assoluti (senza considerare i segni).

34. scrivere un programma che dati due vettori ordinati in modo crescente, ne costruisca un terzo che sia la fusione ordinata in modo crescente dei vettori dati.

35. scrivere un programma che ordini alfabeticamente un vettore contenente stringhe.

36. scrivere un programma che dati due vettori numerici di identiche dimensioni (n(x) e m(x)), costruisca una matrice quadrata (a(x,x)), in cui:

a(i,j)=n(i)*m(j)

37. scrivere un programma che data la matrice quadrata a(x,x), costruisca un'altra matrice quadrata b(x,x) trasposta della prima, cioé in cui:

b(i,j)=a(j,i)

38. scrivere un programma che calcoli il prodotto della matrice quadrata a(xx) per il vettore b(x). Il risultato di tale prodotto è un vettore c(x) cosi definito:

$$c(i) = \sum_{j=1}^{x} a(i,j) * b(j)$$

CAPITOLO 12

ULTERIORI ISTRUZIONI BASIC

Le Istruzioni END, STOP, CONT

COMMODORE–M20: **END**

Questa istruzione termina il programma in esecuzione.

L'istruzione END può essere disposta ovunque nel programma; diversamente dall'istruzione STOP, non manda nessun messaggio di BREAK, ovvero non indica in quale numero di linea è avvenuta l'interruzione. Una istruzione END alla fine del programma è opzionale.

COMMODORE–M20: **STOP**

È simile alla END, ma quando viene eseguita, il programma termina e visualizza:

BREAK IN LINE nnnnn

dove nnnnn è il numero di linea dell'istruzione STOP.

COMMODORE–M20: **CONT**

Per continuare l'esecuzione del programma dopo che è stato premuto il tasto 'STOP', o è stata incontrata una istruzione STOP o una istruzione END. L'esecuzione rincomincia dal punto in cui c'era stata l'interruzione.

CONT è solitamente usata insieme a STOP per il "debugging": quando l'esecuzione è fermata si possono esaminare i dati intermedi. L'esecuzione quindi può ripartire con il comando CONT, oppure con il comando GOTO /nnnnn/, il quale fa ripartire il programma dalla linea /nnnnn/.

Le funzioni TAB e SPC

COMMODORE—M20: **TAB(n)**

Il cursore si posiziona sulla colonna 'n' del video. Se la corrente posizione supera già la n-esima colonna, il cursore si posiziona, su quella colonna, nella riga successiva.

n=0 è la posizione più a sinistra; la larghezza del video meno 1 è la posizione più a destra. TAB può essere usata solo nell'istruzione PRINT (ed anche nel l'LPRINT sull'M20). Nell'M20 la posizione più a sinistra è n=1.

COMMODORE—M20: **SPC(N)**

Stampa 'n' blanks sul video, partendo dall'attuale posizione del cursore. Può essere usata solo nella PRINT ed 'n' deve essere nel range 0, 255.

Definizione di funzioni

COMMODORE—M20:

DEF FN /nome/ (* (/parametro/) *)=
=/definizione di funzione/

Definisce e battezza una funzione scritta dall'utente.

/nome/ deve essere un nome di variabile. Questo nome, preceduto da FN, diventa il nome della funzione.

/parametro/ se non è omesso è il nome della variabile che deve essere rimpiazzata quando la funzione è chiamata.

/definizione di funzione/ è una espressione che specifica le operazioni della funzione. Essa deve essere limitata ad una linea di programma.

I nomi di variabili che appaiono in questa espressione servono solo per la definizione della funzione; essi non influenzano le variabili del programma che hanno lo stesso nome.

Se una variabile usata nella definizione della funzione non appare come parametro, verrà usato il suo valore corrente.

Non sono permesse definizioni di funzioni di tipo stringa.

Se è specificato un tipo nel nome della funzione, il valore dell'espressione è forzato ad essere di quel tipo, prima del ritorno.

Se il tipo specificato nel nome non è associabile con il tipo della variabile usata nella istruzione di chiamata, c'è l'errore:

TYPE MISMATCH

Una istruzione di definizione deve essere eseguita prima della chiamata a tale istruzione, altrimenti avviene l'errore:

UNDEFINED FUNCTION

Nel computer M20, è possibile inoltre definire funzioni con più parametri, ed anche associare un tipo non numerico al nome della funzione.

ESEMPIO

31. definizione e chiamata di funzione:

```
400  REM PARABOLA
410  DEFFNQ(X)=(2*X*X-3*X+7)
420  FOR x=1 TO 20 STEP .5
430  y=FNQ(x)
440  ? x,y
450  NEXT x
460  END
```

ESEMPIO

32. questa funzione restituisce l'arrotondamento matematico del parametro fornito:

```
10  DEFFNAR(Z)=INT(Z+0.5)
```

Le funzioni POS, orologio, USR

COMMODORE–M20: **POS(n)**

Nel computer Commodore, questa funzione restituisce la corrente posizione del cursore lungo l'ascissa del video. La posizione più a sinistra è 0; l'argomento X è solo di comodo.

Nell'M20, restituisce la posizione del cursore lungo l'ascissa se l'argomento è uguale a 0, altrimenti restituisce la posizione del cursore secondo l'ordinata.

Funzioni Orologio TI e TI$ (COMMODORE)

TI è usata per leggere il clock (orologio) interno, e restituisce un valore in 1/60 di secondo.

TI$ è usata per leggere il clock interno e restituisce una stringa di sei caratteri che corrispondono all'ora, ai minuti ed ai secondi. Può essere usata come una comune variabile di stringa, purchè nelle assegnazioni si rispetti il formato descritto (vedi anche esempio 11).

ESEMPIO:

```
50 TI$="091504"
```

per inizializzare l'orologio alle 9.15 e 04 secondi.

Le funzioni Orologio e data (M20)
TIME$, DATE$

Queste funzioni restituiscono l'ora e la data precedentemente inizializzate tramite assegnamento.

ESEMPIO

```
125  TIME$="09.15.04"
128  DATE$="12/07/82"
```

Si noti che il simbolo utilizzato per separare le parti di ora e di data, sarà utilizzato anche nella visualizzazione.

La funzione USR

USR (/espr.num./) (COMMODORE)

USR (* /cifra/ *) (/espr.numer./) (M20)

Questa funzione chiama una subroutine in linguaggio assembler, passandogli come parametro il valore dell'espressione numerica /espr.num./.

ESEMPIO:

```
100  A=Z*COS(T)
110  B=USR(A)
120  C=USR(B+A)
```

Nell'M20 la parte /cifra/ specifica a quale definizione di USR si intende fare riferimento (vedi DEF USR); se è omesso come default si ha USR0. Non si deve inserire alcuno spazio tra USR e /cifra/.

Il Comando POKE e la Funzione PEEK

COMMODORE: **POKE I,J**

Questo comando è utilizzato per scrivere in un byte della memoria.

L'espressione intera I è la locazione di memoria. L'espressione intera J è il dato da memorizzare. J deve essere nel range 0, 255, e I deve essere nel range 0, 65535. La funzione complementare della POKE è la PEEK, ed entrambe sono spesso usate per passare efficientemente argomenti a subroutine in linguaggio assembler.

Inoltre è utilizzata nelle operazioni di OVERLAY (aggancio automatico di programmi — vedi oltre).

Per utilizzare il comando POKE al massimo della sua potenzialità occorrerebbe conoscere molto bene il sistema operativo del Commodore (descritto in testi specifici). Per tale motivo lo tratteremo in modo non esaustivo, con esempi particolari ed essenzialmente associati alla gestione del video.

Al video dei computers Commodore è associata una memoria RAM interna; ogni byte di tale memoria è associato ad una particolare posizione del video: in particolare nei modelli della serie 8000, alla posizione in alto a sinistra è associato il byte 32768 e alla posizione in basso a destra il byte 34767; i byte intermedi corrispondono alle posizioni intermedie sul video.

Nei computers delle serie 3000/4000 il range della memoria associata al video è 32768, 33767.

ESEMPIO

33. nei computer della serie CBM 3000 e CBM 4000, normalmente la tastiera è inizializzata per la scrittura maiuscola e la scrittura dei grafici (shiftando i tasti). È possibile comunque passare nello 'stato carat-

teri' in cui i tasti rispondono con lettere minuscole e maiuscole. Ciò è fatto memorizzando nella locazione di memoria 59468 il numero 14:

```
POKE 59468,14
```

Per ritornare nello stato grafico (lettere maiuscole e grafici), si deve digitare il comando:

```
POKE 59468,12
```

Nei computer della serie CBM 8000 il passaggio dallo stato caratteri allo stato grafico e fatto digitando il comando:

```
PRINT CHR$(142)
```

Per ritornare nello stato caratteri si deve digitare:

```
PRINT CHR$(14)
```

ESEMPIO

34. sfruttiamo la corrispondenza memoria/video per scrivere sul video.
Si usi il programma 34.a. nel caso del computer 8000, e il programma 34.b. nel caso di computer 3000/4000:

```
10  REM   34.A. SERIE 8000
20  POKE 33602,67
30  POKE 33603,73
40  POKE 33604,65
50  POKE 33605,79
60  END

10  REM 34.B. SERIE 3000/4000
15  POKE 59468,14
20  POKE 33140,67
30  POKE 33141,73
40  POKE 33142,65
50  POKE 33143,79
60  END
```

L'istruzione 15 del secondo programma serve a cambiare il set di caratteri.

PEEK(I)

Questa funzione restituisce il valore in decimale del byte letto nella locazione specificata da I: I deve essere nel range 0, 65535.

ESEMPIO

Dopo aver utilizzato i programmi precedenti, si può rileggere la prima lettera scritta sul video utilizzando rispettivamente le istruzioni

```
?PEEK(33140)   serie 3000/4000
?PEEK(33602)   serie 8000
```

CAPITOLO 13

ALTRE ISTRUZIONI BASIC SULL'M20

Definizione di tipi: DEFINT, DEFSNG, DEFDBL, DEFSTR

```
DEFINT  /range/ (* , /range/ *)        :def. intera
DEFSGN /range/ (* , /range/ *)         :def. sing. prec.
DEFDBL /range/ (* , /range/ *)         :def. doppia prec.
DEFSTR /range/ (* , /range/ *)         :def. stringa
```

dove /range/ = /lettera/
oppure /range/ = /lettera/ – /lettera/

Con queste definizioni si specifica alla macchina che tutte le variabili inizianti con una particolare lettera, dovranno essere considerate di un particolare tipo. Se per esempio scriviamo:

```
DEFINT A
```

tutte le variabili con la prima lettera uguale ad A, saranno variabili intere. Se scriviamo:

```
DEFSTR G–K
```

tutte le variabili che iniziano per G, H, I, J, K, saranno di tipo stringa.

Definizione DEF USR

DEF USR (* /cifra/ *) = /espressione num./

È utilizzato per specificare l'indirizzo di inizio di una subroutine in linguaggio macchina.

La cifra rappresenta il numero di sottoprogramma USR che si vuole definire; se viene omessa, come default si ha la cifra '0'.

Il risultato dell'espressione (arrotondato all'intero) è l'indirizzo della subroutine.

Istruzione VARPTR

1. **VARPTR (/variabile/)**
2. **VARPTR (# /numero di file/)**

Nel primo formato questa funzione restituisce l'indirizzo del primo byte associato al dato specificato dalla variabile.

Nel secondo formato se il file è sequenziale, restituisce l'indirizzo di inizio del buffer di I/O associato al file; se il file è relativo, restituisce l'indirizzo del buffer di Field.

Istruzioni di conversione di tipi

CDBL (/espressione numerica/)

Converte il risultato dell'espressione numerica in doppia precisione.

ESEMPIO

```
A   = CDBL(AX)
```

CINT (/espressione numerica/)

Converte il risultato dell'espressione numerica nell'intero più vicino.

ESEMPIO:

```
A% = CDBL(52.58)
? A%
53
OK.
```

CSNG (/espressione numerica/)

Converte il risultato dell'espressione numerica in singola precisione.

ESEMPIO:

```
AA = CSNG(125.231526 # )
? AA
125.231
OK.
```

FIX (/espressione numerica/)
Restituisce la parte intera troncata del risultato dell'espressione.

ESEMPIO:

```
? FIX (52.26), FIX (–52.26)
52      –52
OK.
```

HEX$ (/espressione numerica/)
Converte il risultato dell'espressione numerica (arrotondato all'intero) in una stringa con i caratteri esadecimali corrispondenti al numero.

ESEMPIO:

```
X = 16
? X, HEX$(X)
16      10
OK.
```

OCT$ (/espressione numerica/)
Converte il risultato dell'espressione numerica (arrotondato all'intero) in una stringa con i caratteri ottali corrispondenti al numero.

ESEMPIO:

```
X = 8
? X, OCT$(X)
8      10
OK.
```

Istruzioni per la gestione di stringhe

INSTR (/pos. di inizio/,/stringa/,/sottostringa/)
Ricerca nella stringa specificata, a partire dalla posizione di inizio specifi-

cata, la prima occorrenza della sottostringa; restituisce la posizione trovata, oppure il numero 0 se la sottostringa non è trovata.

SPACE$ (/espressione numerica/)

Restituisce una stringa di spazi con lunghezza specificata dall'espressione.

STRING$ (/lunghezza/ , /espressione numerica/)

STRING$ (/lunghezza/ , /stringa/)

Nel primo caso restituisce una stringa di lunghezza specificata, con caratteri tutti uguali e il cui codice ASCII è specificato dall'espressione numerica.

Nel primo caso restituisce una stringa di lunghezza specificata, con caratteri tutti uguali al primo carattere della stringa specificata.

INPUT$ (/lunghezza/ (* , (* # *) /numero di file *))

legge una stringa di lunghezza specificata dal buffer di tastiera o dal file identificato dal numero di file, senza che i caratteri di controllo (RETURN, punteggiatura, ecc.) abbiano influenza. Il carattere CTRL C comunque interrompe l'esecuzione dell'istruzione.

SWAP /variabile/, /variabile/

Scambia fra loro il contenuto delle variabili specificate. Le variabili devono essere dello stesso tipo e devono effettivamente contenere dati.

ESEMPIO:

```
A=12:B=25:A$="AAA":B$="CCC"
? A,B,A$,B$
SWAP A,B:SWAP A$,B$
? A,B,A$,B$

12      25      AAA     BBB
25      12      BBB     AAA
```

WHILE/WEND

```
WHILE /condizione/
WEND
```

Questa coppia di comandi permette l'esecuzione delle istruzioni comprese fra WHILE e WEND (while end) per tutto il tempo in cui la condizione (espressione logica o relazionale) risulta vera.

Se la condizione è una espressione numerica le istruzioni racchiuse fra WHILE e WEND sono eseguite finché essa è diversa da 0.

Se l'espressione numerica è uguale a 0 o l'espressione logica-relazionale è falsa l'esecuzione prosegue con l'istruzione successiva a WEND.
WHILE-WEND possono essere annidate.

Debugging.

Nel computer M20 sono presenti alcune istruzioni atte alla ricerca ed eventuale gestione degli errori presenti in un programma.

TRON
TROFF

Questi due comandi attivano (TRON = TRACE ON) o disattivano (TROFF = TRACE OFF) lo stato di trace (traccia). Se questo stato viene attivato, si visualizzano man mano che sono eseguite, le istruzioni del programma. Le istruzioni compaiono racchiuse fra parentesi quadre.

ON ERROR GOTO /num.linea/

Durante una esecuzione normale di un programma, il verificarsi di un errore provoca l'interruzione del programma e l'emissione di un messaggio esplicativo. Se è abilitata la routine di gestione dell'errore, il programma non si interrompe ma continua l'esecuzione con la linea specificata dal comando ON ERROR GOTO.
Questo comando abilita la routine di gestione dell'errore, e specifica il numero di inizio della routine.
ON ERROR GOTO 0 disabilita la routine di gestione dell'errore.

ESEMPIO:

```
100 ON ERROR GOTO 1000
110 INPUT # 1, A$
120 ...
1000 REM GESTIONE ERRORE
1010 IF ERR = 15 THEN ...
1020 ...
```

L'errore 15 è 'string too long' (stringa troppo lunga).

ERROR /espressione numerica/

Simula il verificarsi di un errore; l'espressione numerica è il codice d'errore, e deve essere compreso tra 0 e 255.
È usata solitamente per testare la routine di gestione dell'errore e in particolari controlli di dati.

ESEMPIO:

```
100 ON ERROR GOTO 1000
110 INPUT A
120 IF A>800 THEN ERROR 250
...
1000 ?"IL MASSIMO VALORE È 800"
1010 RESUME 110
```

ERR

ERL

Queste variabili contengono, al verificarsi di un errore, il codice di errore (ERR) e il numero della linea in cui l'errore si è verificato (ERL).

Sono solitamente utilizzati, in istruzioni IF...THEN e IF...GOTO, nelle routine di gestione dell'errore.

ESEMPIO:

```
IF ERR = 15 THEN ...
IF ERL = 425 THEN ...
```

Se il comando che causa l'errore è scritto in modo immediato, la variabile ERL specifica il valore 65535.

ESEMPIO:

```
IF 65535 = ERL THEN ...
```

Se il numero di linea specificato con ERL non compare nella parte destra del simbolo '=', non può essere cambiato dal comando di RENUM.

RESUME (* 0 ! NEXT ! /num. linea/ *)

Questo comando permette la continuazione del programma dopo il verificarsi di un errore. Compare in generale, nella routine di gestione dell'errore.

RESUME o RESUME 0, provocano la continuazione della linea che ha provocato l'errore.

RESUME NEXT provoca la continuazione dalla linea successiva a quella dell'errore.

RESUME /num. linea/ provoca la continuazione della linea specificata.

Grafica

Nel computer M20 esiste un insieme di potenti istruzioni per la gestione della grafica.

Il video è composto da una matrice di punti di 512 o 480 punti orizzontali

(secondo il tipo), e da 256 punti verticali. Il video può essere in bianco e nero oppure a colori; in questo contesto noi tratteremo essenzialmente il video bianco e nero.

WINDOW

È possibile dividere il video in un certo numero (al massimo 16) di aree rettangolari dette finestre (window). È possibile trattare ogni finestra come un video particolare ed indipendente dalle altre finestre. Ogni nuova finestra è generata come suddivisione di un'altra finestra (che sarà detta finestra sorgente).

/var/ = WINDOW (/quadrante/,/posizione/ (* ,
(* /spazio verticale/ *) (* ,/spazio orizzontale/ *) *))

/var/ è una variabile intera alla quale il sistema assegna progressivamente un numero (associato quindi alla finestra), da 1 a 16.

L'intero video è considerato come prima finestra quindi con codice 1.

/quadrante/ specifica in quale parte della finestra sorgente si desidera aprire la nuova finestra. Sono permessi i seguenti 4 codici:

0 — parte più alta della finestra sorgente.
1 — parte più bassa della finestra sorgente.
2 — parte più a sinistra della finestra sorgente.
3 — parte più a destra della finestra sorgente.

/posizione/ specifica dove la finestra sorgente deve essere divisa per aprire la nuova finestra.

/spazio verticale/ è parametro opzionale che definisce lo spazio tra le righe: può variare nell'intervallo 10 (si ottiene 25 linee sullo schermo intero) e 16 (si ottiene 16 linee sullo schermo intero).

/spazio orizzontale/ è un parametro opzionale che può assumere solo due valori: il numero 6 (per ottenere 80 caratteri per linea sullo schermo intero) o il numero 8 (per ottenere 64 caratteri sullo schermo intero).

Come caso particolare si ha:

/var/ = WINDOW (0,0 (* , (* /spazio verticale/ *)
(* ,/spazio orizzontale/ *) *))

Con questa definizione non viene aperta nessuna nuova finestra, ma si utilizza per definire l'interlinea e l'interspazio tra i caratteri sullo schermo; al-

l'accensione della macchina (ambiente basic) il video è settato a 16 linee per 64 colonne.

WINDOW % /numero di finestra/

specifica una particolare finestra aperta precedentemente.

COLOR = /codice colore/,/codice colore/,
/codice colore/,/codice colore/

Questa istruzione seleziona 4 degli 8 colori disponibili.

CODICE COLORE	COLORE
0	nero
1	verde
2	rosso
3	giallo
4	blu
5	indaco
6	viola
7	bianco

COLOR (* % /numero di finestra/ , *)
/colore di primo piano/ (* , /colore di sfondo/ *)

definisce il colore di disegno e di sfondo della particolare finestra specificata.

CLS (* % /numero di finestra/ *)

cancella o l'intero video o la finestra specificata.

SCALE (* % /num. fin./ , *) /x0/ , /x1/ , /y0/ , /y1/

Definisce la scala fra il problema dell'utente e le coordinate hardware di video.

x0,x1,y0,y1 sono i vertici della finestra specificata, come desiderati dal problema; sarà poi il sistema a tradurre i disegni in coordinate hardware.

SCALEX e SCALEY (/coordinate/)

Definisce la scala fra il problema dell'utente e le coordinate hardware di video solamente sull'ascissa o sull'ordinata.

CLOSE WINDOW (* % /num. fin./ *)

Chiude la finestra specificata o tutte le finestre se non compare la parte opzionale.

CURSOR (* POINT *) (* (X,Y) *) (* on/off
(* ,/lamp./ (* ,/forma/ *) *) *)

Questo comando definisce le coordinate, specificate da X e Y, in cui il cursore per caratteri (CURSOR) e il cursore per grafici (CURSOR POINT) deve posizionarsi.

on/off specifica se il cursore deve (on/off = 1) o non deve (on/off = 0) visualizzarsi.

/lamp./ specifica se il cursore non deve lampeggiare (lamp.=0) o deve lampeggiare con una intermittenza di x pulsazioni al secondo; x deve essere compreso nell'intervallo 1, 20.

/forma/ è un campo opzionale che altera l'aspetto del cursore. È un array di interi definito dall'utente e che definisce i pixel del cursore che devono essere accesi.

PSET (* % /num.fin./ *) (X,Y) (* , /colore/ *)

Accende il punto (pixel) specificato da X,Y, del colore di /colore/ se specificato, altrimenti del colore di primo piano.

PRESET (* % /num.fin./ *) (X,Y)

Rimette il pixel specificato al colore dello sfondo.

PAINT (* % /num.fin./, *) (X,Y) (* ,/colore/
(* ,/colore bordo/ *) *)

Colora con il colore specificato la figura chiusa che raccoglie il pixel di coordinate X, Y. Se specificato /colore bordo/ viene colorato anche il bordo della figura.

/var./ = POINT (X,Y)

Restituisce il colore attivato nel punto di coordinate X, Y.

LINE (* % /num.fin./ *) (* (* STEP *) (X1,Y1) – *)
(* (* STEP *) (X2,Y2) *) (* ,(* /colore/ *)
(* ,(* B (* F *) *) (* ,/azione/ *) *) *)

Permette di tracciare una linea od un rettangolo sullo schermo.

STEP è una parola chiave opzionale; se specificata permette di definire coordinate relative all'ultimo pixel disegnato sullo schermo.

X1,Y1 sono le coordinate del primo punto della linea o un vertice del rettangolo; se sono omessi, il punto di inizio è preso l'ultimo pixel disegnato o il punto di coordina 0,0 (il vertice inferiore sinistro della finestra considerata).

X2,Y2 sono le coordinate dell'ultimo punto sulla linea o il vertice opposto del rettangolo.

/colore/ specifica il colore della linea o del rettangolo disegnato; se non è specificato, viene utilizzato il colore di primo piano.

B (box) è un parametro opzionale che se specificato permette di tracciare il rettangolo con lati paralleli alla finestra e con diagonale specificata dalla coppia di coordinate.

F (filled) può essere specificato solo insieme a B: BF traccia un rettangolo completamente colorato.

/azione/ può assumere uno dei seguenti valori: AND, OR, XOR, NOT, PSET, PRESET. PSET specifica che la linea o il rettangolo o il rettangolo riempito devono essere disegnati con il particolare colore (/colore/); PRESET indica invece che la linea o il rettangolo o il rettangolo riempito devono essere disegnati con il colore di sfondo; AND, OR, XOR specificano che la linea o il rettangolo o il rettangolo riempito saranno tracciati con il colore risultate dall'operazione logica applicata al colore esistente sullo schermo e quello specificato; NOT inverte il colore esistente sullo schermo. Come default il sistema considera l'azione PSET.

CIRCLE (* % /num.fin./, *) (X,Y) ,/r/ (* ,(* ,/colore/ *)
(* , (* /rapp./ *) (* ,/azione/ *) *) *)

Disegna un cerchio o un ellisse sul video.

Il centro del cerchio è specificato dalle coordinate X,Y; il raggio è specificato dal parametro /r/.

Il centro dell'ellisse è specificato da X,Y; in questo caso /r/ rappresenta la dimensione di mezzo asse orizzontale; la dimensione dell'asse verticale è specificata da /rapp./ che rappresenta il rapporto assiale dell'ellisse. /rapp./ deve essere un numero positivo; come default si ha /rapp./=0.85 che, a causa della diversa densità di pixel tra orizzontale e verticale, genera un cerchio.

/azione/ definisce il colore di ogni punto sulla curva con le stesse modalità viste nell'istruzione LINE.

Memorizzazione di disegni.

È possibile memorizzare parte o tutta una finestra in un vettore di interi, o al contrario, visualizzare in una certa parte di video, un disegno precedentemente memorizzato in un vettore di interi.

GET (* % /num.fin./ *) (X1,Y1)–(X2,Y2) ,/array/

Memorizza nell'array unidimensionale /array,/ l'immagine dello schermo compresa nel rettangolo di vertici opposti X1,Y1 e X2,Y2. L'array, che viene specificato tramite il suo primo elemento, deve essere precedentemente dimensionato tramite la seguente formula:

video in bianco e nero: $\frac{vp * op}{16} + 3$

video a colori: $\frac{vp * op}{8} + 3$

dove hp = dimensione verticale in pixel
op = dimensione orizzontale in pixel

PUT (* % /num.fin./ *) (X1,Y1)–(X2,Y2) ,/array/ (* ,/azione/ *)

Visualizza l'immagine precedentemente memorizzata nell'array specificato, nel rettangolo di vertici opposti X1,Y1 e X2,Y2.
/azione/ definisce il colore di ogni punto dell'immagine con le stesse modalità viste nell'istruzione LINE.

DRAW (* % /num.fin./ *) /stringa di comando/

Muove un pennello virtuale all'interno della finestra, e traccia linee di un certo colore.
La stringa di comando è formata con i comandi specificati nella seguente tabella:

- M dx,dy :sposta il cursore dalla posizione attuale (x,y) alla posizione (x+dx,y+dy).
- J x,y :sposta il cursore alla posizione di coordinate x,y.
- U dy :sposta il cursore in sù di 'dy' pixel.
- D dy :sposta il cursore in giù di 'dy' pixel.
- L dx :sposta il cursore a sinistra di 'dx' pixel.
- R dx :sposta il cursore a destra di 'dx' pixel.
- C /colore/ :attiva un particolare colore per il pennello.

Ogni comando può essere preceduto dall'opzione B, che inibisce la traccia del pennello; e può essere seguito dal codice di azione che permette di definire (come si è specificato nel comando LINE) il tipo di colore tracciato sul video.

I verbi dell'azione sono rappresentati con le lettere:

A	AND
O	OR
X	XOR
N	NOT
P	PSET
R	PRESET

Istruzioni di sistema

SYSTEM

Questo comando permette di ritornare nell'ambiente di sistema operativo PCOS. Può essere utilizzato sia in programma che in modo diretto. Chiude tutti i files dati aperti.

EXEC /espressione stringa/

Esegue un comando PCOS o una subroutine in assembler. /espressione stringa/ é interpretato come un nome di subroutine seguito da una serie di argomenti costanti. Se EXEC esegue un comando PCOS, il contenuto della stringa deve essere esattamente quello che si scriverebbe per lo stesso comando in ambiente PCOS.

ESEMPIO:

```
10  EXEC· "pl vc"
20  A$="vc 1 :=0:"
30  EXEC A$
```

Questa subroutine permette la stampa di lettere minuscole:

```
60000 R8$="":L9=LEN(R9$):FORW9=1TOL9:
C9$=MID$(R9$,W9,1) :A9=ASC(C9$)
60010 IFA9>64ANDA9<91 THENA9=A9+32
60020 R8$=R8$+CHR$(A9) :NEXTTW9:RETURN
```

La riga da stampare perviene alla subroutine come R9$ e ritorna alla routine principale per la stampa su stampante Honeywell come R8$.

Esempio di programma per Subroutine

```
 5  POKE 59468, 14
10  INPUT "RIGA"; R9$:GOSUB60000
20  OPEN 1,5: PRINT#1,R8$: CLOSE 1
40  END
```

APPENDICE C

Per il sistema Commodore, oltre alle caratteristiche di sistema analizzate in questo testo, sono stati sviluppati numerosi supporti hardware/software che potenziano notevolmente le sue possibilità. In questa appendice descriveremo brevemente alcuni tra i più interessanti 'optional' disponibili.

1. SCHEDA GRAFICA AD ALTA RISOLUZIONE

Questa scheda, sviluppata dalla Commodore U.K. propone le seguenti possibilità:

- una pagina di video con risoluzione 512 x 512
- due pagine di video con risoluzione 512 x 256
- scrolling tra le due pagine di video
- commutazione tra le pagine di video
- rappresentazione di entrambe le pagine sovrapposte
- rappresentazione contemporanea o separata delle normali istruzioni di video e di grafica
- labels su video in corrispondenza di punti o linee, a partire da qualsiasi posizione e nella direzione dei quattro assi maggiori.
- gestione diretta del cursore per disegnare su video da tastiera
- disegno sotto modalità di reverse

Segue una lista dei comandi che si vanno a sommare a quelli già esistenti:

GRMODE: stabilisce le sovrapposizioni video e le modalità di risoluzione

CLEAR: cancella il contenuto delle pagine di video

PAGE: definisce la pagina su cui lavorare e la pagina da visualizzare (non necessariamente coincidenti)

SCROLL: permette lo scorrimento del video verso l'alto o verso il basso

REV: attiva il reverse

MOVE: trasferisce al punto con coordinate specificate, la posizione del disegno

IMOVE: trasferimento relativo della posizione del disegno

DRAW: traccia una linea dalla posizione occupata sino alla posizione specificata; può essere visibile, non visibile o reverse

IDRAW: traccia una linea in modo relativo

PEN: definisce il modo di disegnare; può essere normale, non visibile, reverse, per cancellare

LANGLE: definisce un valore di rotazione

APT: trasferisce la posizione alle coordinate specificate

IPT: trasferisce la posizione in modo relativo

FILL: riempie un'area rettangolare, con diverse modalità

LABEL: scrive una stringa nella posizione attualmente occupata

LPLACE: comanda il posizionamento della label

WHERE: riporta nell'argomento le attuali posizioni del disegno

TESTP: esamina lo stato del pixel alle coordinate specificate

CURS: visualizza il cursore mentre disegna

OBJECT: visualizza un oggetto alle coordinate specificate

RBYTE: legge un byte dalla memoria video ad un indirizzo specificato

WBYTE: scrive un byte nella memoria video

VDU: dispone il PET in emulazione VDU

2. SuperKRAM

Il SuperKRAM (metodo di accesso random a chiave) è un potente sistema operativo che permette la gestione generalizzata di dati afferiti tramite chiavi alfanumeriche. L'origine teorica di questo S.O. è il calcolo relazionale utilizzato nella scrittura dei DBMS relazionali (Data Base Management System: sistema per la gestione di banche di dati). La sua architettura è molto efficiente e permette una riorganizzazione dinamica della struttura ad indici, permettendo quindi un minimo tempo di ricerca.

Per metodo di accesso a chiave alfanumerica si intende che il gruppo di dati riferiti come record, sono indirizzati tramite un nome anzichè il numero di posizione, come invece accade nel metodo di accesso random.

Nel superKRAM possono esistere più chiavi per afferire un record: una di queste, con la caratteristica necessaria di essere unica per ogni record dell'archivio, è detta chiave primaria; tutte le altre eventuali chiavi sono dette chiavi alternative. Una chiave alternativa in generale non indirizza univocamente un record ma tutti i record con tale chiave coincidente; dovrà quindi essere fatta una selezione all'interno del gruppo riferito, per determinarne il record desiderato. Per esempio se organizziamo una rubrica telefonica, il numero telefonico sarà la chiave primaria (in quanto é sicuramente unico per ogni persona presente nell'archivio), ed il cognome potrebbe essere una chiave secondaria, utilizzata per esempio nella ricerca del record desiderato: naturalmente potranno esistere più persone con lo stesso cognome, quindi si dovrà cercare quella interessata tra tutte quelle restituite dalla macchina.

Segue una descrizione sintetica delle istruzioni messe a disposizione dal Super KRAM (oltre a quelle già esistenti).

I) Funzioni per la gestione del sistema:

UBUF: questo comando é utilizzato per definire il numero di archivi (file dati) che possono essere contemporaneamente aperti (possono essere al massimo 10). UBUF genera lo spazio necessario nell'area adibita alla gestione del sistema (User Buffer Pool e Working Storage).

INIT: cancella l'area User Buffer Pool e Working Storage.

KILL: rimuove il SuperKRAM dalla partecipazione attiva nel sistema.

II) Funzioni per la gestione dei File:

CREABASE: questo comando è utilizzato per creare inizialmente un file di dati.

CREAIX: questo comando è utilizzato per definire chiavi alternative.

ACCESS: questo comando è utilizzato per aprire un file di dati precedentemente creato.

SHUT: questo comando è utilizzato per chiudere un file di dati.

III) Funzioni per la gestione di I/O (ingresso/uscita):

GTREC: questo comando permette di leggere un record da un file di dati.

RDREC: questo comando permette di leggere in maniera sequenziale, records da un file di dati.

RDSET: questo comando permette di leggere in modo sequenziale, records con chiave alternativa identica.

PUT: questo comando é utilizzato per variare un record esistente.

ADD: questo comando è utilizzato per aggiungere un record al file di dati.

ERASE: questo comando è utilizzato per eliminare un record dal file di dati.

3. SCREEN GENERATOR EDITOR

Lo Screen generator editor consente l'utilizzo di un nuovo set di istruzioni BASIC che permettono all'utente una più potente gestione di video.

In particolare è possibile:

— tracciare linee orizzontali e verticali.
— visualizzare una variabile in una certa posizione.
— salvare e caricare una pagina di video da disco.
— definire campi per operazioni di data-entry.
— definire il formato dei dati.
— operare in precisione multipla con 22 cifre significative ed esponente variabile tra —64 e +63

Segue una sintetica descrizione delle istruzioni aggiuntive.

A) gestione di video:

LINE: disegna una linea orizzontale sul video.

COLL: disegna una linea verticale sul video.

OUT: visualizza in una posizione specificata e con formato specificato il contenuto di una variabile.

CLEAR: cancella una posizione specificata.

SP: salva una pagina di video sul disco.

LP: carica una pagina di video dal disco.

SCROLL: esegue lo scroll su una parte di video.

B) gestione del data - entry (ingresso dati):

La parte di sistema operativo che gestisce il data-entry (mini-editor) lavora su un campo detto data-entry-filed. Su questo particolare campo il mini-editor esegue la gestione di ingresso-uscita dati e le operazioni in doppia precisione.

REQZ: richiama il mini - editor per la gestione dei dati.

INZ: assegna il valore del data - entry - field ad una variabile.

ERAZ: annulla il data - entry - field.

DECZ: definisce il formato del data - entry - field.

ADD: esegue addizioni in doppia precisione.

SUB: esegue sottrazioni in doppia precisione.

MULT: esegue moltiplicazioni in doppia precisione.

DIV: esegue divisioni in doppia precisione.

APPENDICE D

LISTATI DI PROGRAMMI BASIC SU M20

In questa appendice sono riportati alcuni programmi esemplificativi riportati nel testo, con le opportune modifiche, necessarie per essere eseguiti sull'M20.

I programmi non riportati sono eseguibili direttamente anche sull'M20, con l'unica differenza presente nell'istruzione di pulizia video: nell'M20 si dovrà sostituire l'istruzione 'PRINT /clr-home/' con l'istruzione 'CLS'.

```
10  REM esempio10
20  REM quadrato
25  CLS
30  K$="     ***************************"
35  K1$="     *     *"
40  PRINT : PRINT : PRINT
50  PRINT K$
60  PRINT K1$:PRINT K1$ : PRINT K1$ : PRINT K1$ : PRINT K1$
70  PRINT K1$ : PRINT K1$ : PRINT K1$ : PRINT K1$
80  PRINT K$
```

```
10  REM esempio 11
20  REM orologio
30  CLS
40  INPUT "che ora è (HH/MM/SS)";TI$
45  TIME$=TI$
48  CLS
50  CURSOR (10,5) : PRINT "l'ora è : "TIME$
60  GOTO 50
```

```
10   REM esempio17
20   REM funzione
22   W=WINDOW (0,0,10,6)
25   L1=20
30   INPUT"primo estremo";A
40   INPUT"secondo estremo";B
50   INPUT"passo";C
60   GOSUB 1000:REM disegna quadro
70   MN=SIN (A) :MX=SIN (A)
75   FOR =A TO B STEP C
80   Y=SIN (X)
90   IF Y>MX THEN MX=Y
100  IF Y<MN THEN MN=Y
110  NEXT X
120  PS=(MX-MN)/L1
130  IX=0
140  FOR X=A TO B STEP C
150  IX=IX+1
160  IY=L1-1NT ((SIN (X) - MN)/PS+, 5)
170  GOSUB 2000:REM disegna punto
180  NEXT X
190  GOTO 190
1000 REM disegna quadro
1010 CLS
1020 LINE (0,0) - (512,255) ,, B
1025 LINE (0,128) - (512, 128)
1030 RETURN
2000 REM disegna punto
2010 CURSOR (IX+1, IY+3)
2015 PRINT"*";
2020 RETURN
```

```
10   REM esempio 19
20   REM random
30   CLS
40   INPUT"quanti numeri";N
50   INPUT"minori di";M
60   RANDOMIZE
70   CLS
```

```
  80  FOR I=1 TO N
  90  GOSUB 1000
 100  PRINT X,
 110  NEXT I

 120  END
1000  REM generatore
1010  X=INT(RND*M)
1020  RETURN
```

```
  10  REM esempio20
  20  REM dado
  22  CLS
  25  RANDOMIZE
  30  CLS
  40  X=INT (RND*6+1)
  50  ON X GOSUB 1000,2000,3000,4000,5000,6000
  60  PRINT : PRINT : PRINT
  70  INPUT"altro lancio";X$
  80  IF X$"s" THEN 30
  90  END
1000  GOSUB 9000
1010  GOSUB 9090
1020  RETURN
2000  GOSUB 9010
2010  GOSUB 9020
2030  GOSUB 9090
2040  RETURN
3000  GOSUB 9000
3010  GOSUB 9010
3020  GOSUB 9020
3030  GOSUB 9090
3040  RETURN
4000  GOSUB 9010
4010  GOSUB 9020
4020  GOSUB 9030
4030  GOSUB 9060
4035  GOSUB 9090
4040  RETURN
```

```
5000  GOSUB 9010
5010  GOSUB 9000
5020  GOSUB 9020
5030  GOSUB 9030
5040  GOSUB 9060
5050  GOSUB 9090
5060  RETURN
6000  GOSUB 9010
6010  GOSUB 9020
6020  GOSUB 9030
6030  GOSUB 9040
6040  GOSUB 9050
6050  GOSUB 9060
6060  GOSUB 9090
6070  RETURN
9000  LINE (251,133) – (261,143) ,, BF : RETURN
9010  LINE (221,173) – (231,163) ,, BF : RETURN
9020  LINE (281,103) – (291,113) ,, BF : RETURN
9030  LINE (221,103) – (231,113) ,, BF : RETURN
9040  LINE (221,133) – (231,143) ,, BF : RETURN
9050  LINE (281,133) – (291,143) ,, BF : RETURN
9060  LINE (281,163) – (291,173) ,, BF : RETURN
9090  LINE (201,83) – (311,193) ,, BF,NOT : RETURN
50409  60
```

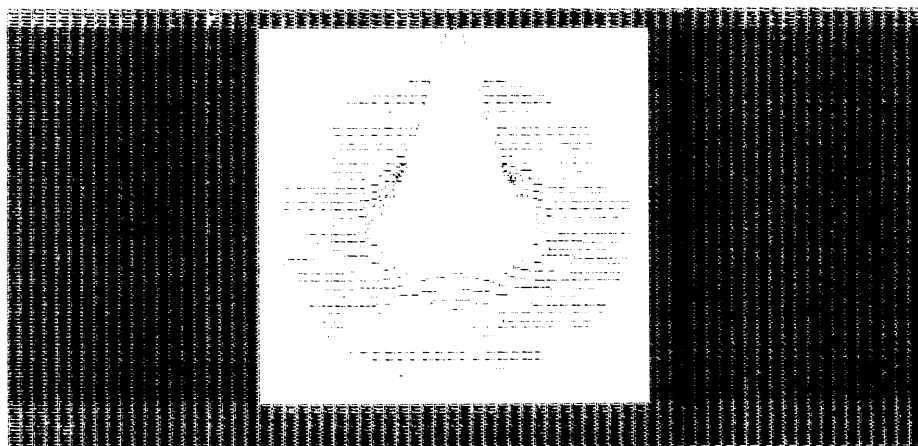
```
10  REM esempio27
20  REM generatore parole
25  RANDOMIZE

30  CLS
40  C$="bcdfglmnprstyz"
50  V$="aeiou"
60  FOR J = 1 TO 100
70  U$=""
80  FOR I=1 TO 3
90  GOSUB 1000
100  A=INT(X*14)+1
110  U$=U$+MID$(C$,A,1)
120  GOSUB 1000
```

```
 130  A=INT (X*5)+1
 140  U$=U$+MID$(V$,A,1)
 150  NEXT I
 160  PRINT U$,
 170  NEXT J
 180  END
1000  REM generatore
1010  X=RND :RETURN
```

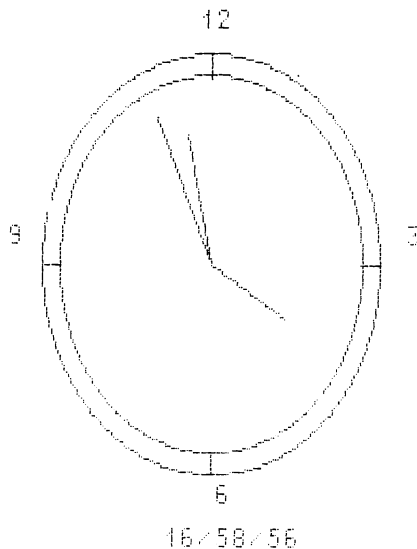
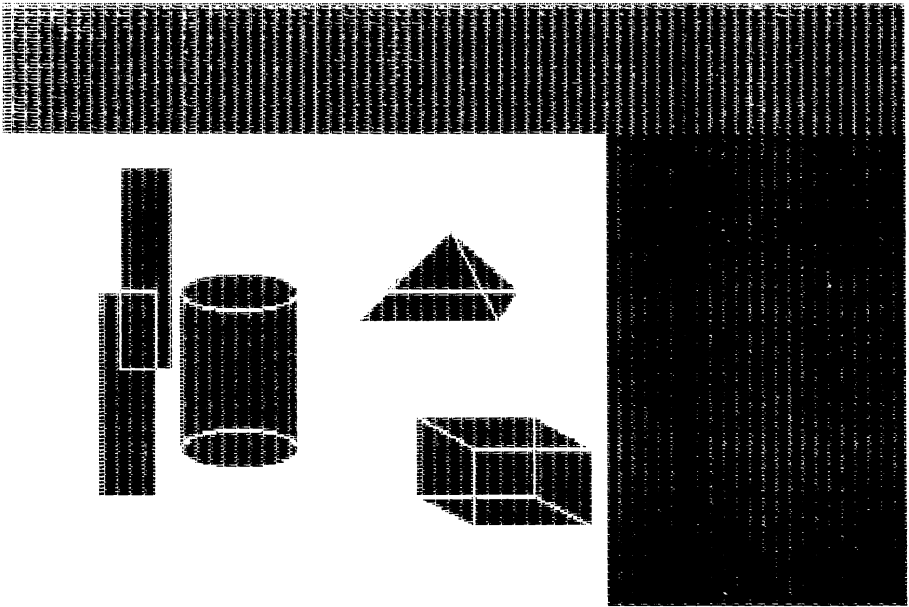
APPENDICE E

ALCUNI PROGRAMMI DIMOSTRATIVI SU M20

GRAFICO TRIDIMENSIONALE by D.R

```
 10  CLS
 20  EXEC"la'GRAFICO TRIDIMENSIONALE by D,R, ,60,200,2,0
 60  DEF FNA(Z)=90*EXP (–Z*Z/600)
 70  K=5
120  FOR X=–100 TO 100
130  L=0 : P=1
160  Y1=K*INT (SQR(10000–X*X)/K)
170  FOR Y=Y1 TO –Y1 STEP –K
190  Z=INT (100+FNA(SQR(X*X+Y*Y)) – ,70106*Y)
220  IF Z<L GOTO 300
230  L=Z
260  PSET(X+250,Z)
280  IF P=0 THEN Z1=Z
290  P=0
```

```
300  NEXT Y,X
330  GOTO 390
360  RETURN
390  LINE(140,20) – (360,190),,B
400  LINE(0,198) – (512,198)
410  PAINT(0,0)
430  EXEC“SP”
```

```
  5  REM orologiom20
 10  PI=3, 1415
 20  CLS
 30  CURSOR(30,5) : PRINT“ora:min:sec”
 40  CURSOR(30,6) : PRINT“/     I     /
 50  CURSOR(31,7):INPUT;“”, A$
 60  TIME$=A$
 65  CLS
 70  CIRCLE(256, 128),70,,1
 80  CIRCLE(256,128),78,,1
 90  LINE(256,198) – (256,206)
100  LINE(256,50) – (256,58)
110  LINE(186,128) – (178,128)
120  LINE(326,128) – (334,128)
130  CURSOR(32,3):PRINT“12”
140  CURSOR(44,8):PRINT“3”
150  CURSOR(33,14):PRINT“6”
160  CURSOR(21,8):PRINT“9”;
170  GOSUB 500:GOSUB 540:GOSUB 580
180  GOSUB 1000
185  LINE(256,128) – (XS,YS),,,NOT
190  IF SI=>MI THEN LINE(256,128) – (XM,YM)
195  IF SI=>OI*5+1–60 THEN LINE(256,128) – (XO,YO)
196  GOSUB 1000
200  GOSUB 580
210  CURSOR(1, 1) :PRINT CHR$(7)
215  IF SI=59 THEN GOSUB 300
220  FOR T=0 TO 750:NEXT T
225  GOTO 185
300  LINE(256,128) – (XM,YM),,,NOT
310  IF MI=>OI*5+1–60 THEN LINE(256,128) – (XO,YO)
```

```
 315   GOSUB 540
 330   IF MI=>59 THEN GOSUB 400
 340   RETURN
 400   LINE(256,60) - (XO,YO) ,,, NOT
 410   OI=VAL (LEFT$(TIME$,2))
 420   GOSUB 500
 430   RETURN
 500   OI=VAL(LEFT$(TIME$,2)):A=OI*PI/6+PI/2
 505   XO=256-40*COS(A)
 510   YO=128+40*SIN(A)
 520   LINE(256,128) - (XO,YO)
 530   RETURN
 540   MI=VAL(MID$(TIME$,4,2)):A=MI*PI/30+PI/2
 545   XM=256-50*(COS(A)
 550   YM=128+50*SIN(A)
 560   LINE(256,128) - (XM,YM)
 570   RETURN
 580   SI=VAL(RIGHT$(TIME$,2)):A=SI*PI/30+PI/2
 585   XS=256-60*COS(A)
 590   YS=128+60*SIN(A)
 600   LINE(256,128) - (XS,YS)
 610   RETURN
1000   CURSOR (30,15):PRINT TIME$
1010   RETURN
```

```
  5  CLS
  6  SCALE 0,768,0,384
 10  CIRCLE (200,200),50,,.25
 20  CIRCLE (200,100),50,,.25
 30  LINE (149,200) – (149,100)
 40  LINE (250,200) – (250,100)
100  LINE (400,50) – (500,100),,B
110  LINE (350,70) – (450,120),,B,NOT
120  LINE (400,50) – (350,70)
130  LINE (400,100) – (350,120)
140  LINE (500,50) – (450,70)
150  LINE (500,100) – (450,120)
155  GOSUB 500
200  PAINT (0,0)
210  LINE (0,0) – (512,300),,BF,NOT
220  GOTO 210
500  LINE (80,70) – (130,200),,B
510  LINE (100,150)–(145,280),,B,NOT
520  LINE  (300,180) – (420,180)
530  LINE  (320,200) – (440,200)
540  LINE (300,180) – (320,200)
```

```
550   LINE (420,180) – (440,200)
560   LINE (300,180) – (380,240)
570   LINE (420,180) – (380,240)
580   LINE (320,200) – (380,240)
590   LINE (440,200) – (380,240)
600   RETURN
```

esempio file random

```
 500  REM ———— apertura file ——————
 510  OPEN "R" , 1, "0:indirizzi", 84
 520  REM ———— definizione campi ————
 530  FIELD # 1,30 AS NOME$, 20 AS INDIRIZZO$, 20 AS CITTA$,
        12 AS TELEFONO$, 2 AS ETA$
1000  REM ——————— main ————————
1020  CLS : PRINT TAB(23)"GESTIONE INDIRIZZI"
1025  PRINT STRING$(64, "–")

1030  PRINT:PRINT:PRINT
1040  PRINT TAB(10)"(1) INSERIMENTO – CORREZIONE"
1050  PRINT
1060  PRINT TAB(10)"(2) VISUALIZZAZIONE"
1070  PRINT
1080  PRINT TAB(10)"(f) FINE"
1090  CURSOR (1,14):PRINT"premi un tasto"
1100  A$=INKEY$:IF A$ *""GOTO 1100
1110  IF A$="1"THEN GOSUB 2000
1120  IF A$="2"THEN GOSUB 3000
1130  IF A$="f"THEN CLOSE:END
1140  GOTO 1020
2000  REM ————— inserimento e variazione —————————
2010  CLS:PRINT TAB(12) "INSERIMENTO VARIAZIONE"
2020  PRINT STRING$(64, "–")
2030  PRINT:PRINT
2040  INPUT"CODICE";CODICE: IF CODICE = 0 THEN RETURN
2050  PRINT
2060  INPUT"NOME";NOMEW$
2070  INPUT"INDIRIZZO";INDIRIZZOW$
2080  INPUT"CITTÀ";CITTAW$
2090  INPUT"TELEFONO";TELEFONOW$
```

```
2100  INPUT"ETÀ  ";ETA%
2110. CURSOR (1,14):PRINT"premere per conferma '/'"
2120  A$ = INKEY$:IF A$="" THEN 2120
2130  IF NOT ( A$ = "/" ) THEN 2000
2140  LSET NOME$ = NOMEW$
2150  LSET INDIRIZZO$ = INDIRIZZOW$
2160  LSET CITTA$ =CITTAW$
2170:  LSET TELEFONO$ = TELEFONOW$
2180  RSET ETA$ = MK1$ (ETA%)
2190  PUT #1, CODICE
2200  GOTO 2000
3000  REM ------ visualizzazione -------
3010  CLS:PRINT TAB(23)"VISUALIZZAZIONE"
3020  PRINT STRING$(64, "-")
3030  PRINT:PRINT
3040  INPUT"CODICE";CODICE: IF CODICE = 0 THEN RETURN
3050  PRINT
3060  GET #,CODICE
3070  ETA% = CUI (ETA$)
3075  PRINT"NOME";NOME$
3080  PRINT"INDIRIZZO";INDIRIZZO$
3090  PRINT"CITTÀ";CITTA$
3100  PRINT"TELETONO";TELEFONO$
3110  PRINT"ETÀ  ";ETA%
3120  CURSOR (1,14):PRINT"premi un tasto per proseguire"
3130  A$ = INKEY$ : IF A$="" THEN 3130
3140  GOTO 3000
```

Alcuni programmi dimostrativi su Commodore con scheda grafica

```
20  PRINT"♥"
22  REM INPUT ORA MIN SEC PARTENZA
23  POKE59467,16:POKE59466,51:POKE59464,0
25  PRINT"]]]]]]]]]]]]]]]]]QQQQQQQORA MIN SEC"
30  PRINT"]]]]]]]]]]]]]]]]]  N   I   /"
35  PRINT"]]]]]]]]]]]]]]]]]  N   I   /"
38  INPUT"]]]]]]]]]]]]]]]]]";T$
40  TI$=T$
45  SYS37602:CLEAR:GRAPH:REM CHIAMATA ROUTINE GRAFI-
    CA,CLEAR PAGINA GRAFICA
```

```
46   GOSUB1100
47   GOSUB650
48   CHAR"12",152,179 : CHAR"3",242,95 : CHAR"6",156,10 :
     CHAR"9",70,95
70   GOSUB500:GOSUB540:GOSUB580
80   GOSUB1000
90   RDRAW160,99,XS,YS
110  IFSI=>MITHENDRAW160,99,XM,YM
120  IFSI=>OI*5+1-60THENDRAW160,99,XO,YO
140  GOSUB580
145  POKE59464,50:FORR=0TO20:NEXTR:POKE59464,0
150  IFSI=59THENGOSUB200
153  GOSUB1000
154  FORT=0TO330:NEXTT
155  GOTO90
200  RDRAW160,99,XM,YM
230  IFMI=>OI*5+1-60THENDRAW160,99,XO,YO
250  GOSUB540
260  IFMI=59THENGOSUB300
270  RETURN
300  RDRAW1160,99,XO,YO
330  OI=VAL(LEFT$(TI$,2))
350  GOSUB500
360  RETURN
500  OI=VAL(LEFT$(TI$,2)):A=OI*π/6+π/2
505  XO=160-40*COS(A)
510  YO=99+40*SIN(A)
520  DRAW160,99,XO,YO
530  RETURN
540  MI=VAL(MID$(TI$,3,2)):A=MI*π/30+π/2
545  XM=160-50*COS(A)
550  YM=99+50*SIN(A)
560  DRAW160,99,XM,YM
570  RETURN
580  SI=VAL(RIGHT$(TI$,2)):A=+SI*π/30+π/2
585  XS=160-60*COS(A)
590  YS=99+60*SIN(A)
600  DRAW160,99,XS,YS
610  RETURN
650  R=70:GOSUB700
```

```
660  R=78:GOSUB700
670  GOSUB750
680  RETURN
700  FORT=0TOπ*2STEPπ/230
710  DRAW160+R*COS(T),99-R*SIN(T)
720  NEXTT
730  RETURN
750  FORT=π/2TO-3/2*πSTEP-π/6
760  A=70*COS(T)+160 : B=99-70*SIN(T) : C=78*COS(T)+160 :
     D=99-78*SIN(T) : DRAWA,B,C.D
770  NEXT
780  RETURN
1000 CHARTI$,250,20
1010 RETURN
1100 FORY=0TO10:FORU=0TO100:NEXTY:CHAR" O R O L O G I O
     DR.",80,190
1200 FORU=0TO100:NEXTU:CHAR"  O  R  O  L  O  G  I  O
     DR.",83,190:NEXTY
1300 RETURN
```

READY

```
 10  SYS37602
 20  CLEAR:GRAPH
 60  DEFFNA(Z)=90*EXP(-Z*Z/600)
110  K=5
120  FORX=-100TO100
130  L=0:P=1
160  Y1=K*INT(SQR(10000-X*X)/K)
170  FORY=Y1TO-Y1STEP-K
190  Z=INT(100+FNA(SQR(X↑2+Y↑2))-.70106*Y)
220  IFZ<LGOTO300
230  L=Z
260  DRAWX+150,Z
280  IFP=0THENZ1=Z
290  P=0
300  NEXTY,X
390  END
READY.
```

APPENDICE F

CODICI ASCII

ASCII CODE	CHARACTER	ASCII CODE	CHARACTER	ASCII CODE	CHARACTER
000	NULL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	—	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	bkslash
007	BEL	050	2	093	]
008	BS	051	3	094	↑
009	HT	052	4	095	back arr
010	LF	053	5	096	space
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	;
038	&	081	Q	124	<
039	'	082	R	125	=
040	(	083	S	126	>
041	)	084	T	127	DEL
042	*	085	U		

ELETTRONICA INTEGRATA DIGITALE

di Erbert Taub e Donald Schilling

Non esiste, in lingua italiana, un libro di testo così. Chiaro, completo, moderno, ma anche rigoroso e didattico. Sono alcuni tra gli aggettivi che costituiscono la prerogativa di questo volume. Per capire l'elettronica digitale bisogna avere delle solide conoscenze sui dispositivi a semiconduttore, soprattutto usati in circuiti di commutazione. E malgrado quest'analisi richieda una notevole complessità matematica, introducendo alcune semplificazioni, è possibile mantenere la trattazione ugualmente rigorosa e ottenere approssimazioni pienamente accettabili. Come trascurare poi gli amplificatori operazionali, che, se a rigore non rientrerebbero nella materia, però trovano larga applicazione in sistemi completamente digitali. E poi i circuiti integrati, finalmente spiegati e analizzati in tutti i loro aspetti. Dalla vecchia logica resistore-transistor (RTL), funzionale nella sua semplicità all'esemplificazione degli aspetti fondamentali, a quella a simmetria complementare (CMOS). Questo, però, dopo aver studiato un capitolo che, pur non richiedendo alcuna conoscenza preliminare, va a fondo dei concetti di variabili logiche, di algebra di Boole, di analisi di circuiti logici. E ancora. Via via nei vari capitoli: i flip-flop, i registri, e i contatori (sia sincroni che asincroni), i circuiti logici atti ad eseguire operazioni matematiche, le memorie a semiconduttore (RAM, ROM, EPROM, ...), l'interfacciamento tra segnali analogici e digitali (multiplexer, circuiti sample and hold, ..., convertitori d/a e a/d), i temporizzatori. Tutto con oltre 400 problemi, dai più semplici ai più sofisticati, in cui vengono presentati i circuiti tipici che si trovano nella pratica.
Un testo quindi non solo per gli specialisti e per gli studenti universitari, ma che si adatta magnificamente agli Istituti Tecnici.
Un testo che, speriamo per gli studenti, la scuola non debba scoprire tra alcuni anni.

SOMMARIO
Dispositivi Elettronici Fondamentali; Amplificatori Operazionali e Comparatori; Circuiti Logici; Logica Resistore-Transistore e Logica ad Iniezione Integrata; Logica Diodo-Transistore; Logica Transistore-Transistore, Logica ad Accoppiamento di Emettitore; Porte MOS; I Flip-Flop; Registri e Contatori; Operazioni Aritmetiche; Memorie a Semiconduttore; Interruttori Analogici; Conversione Analogico-Digitale; Circuiti di Temporizzazione; Linee di Trasmissione; Problemi; Alcuni Esempi di Specifiche.

Pag. 740 Formato 16,5x23 Cod. 204A

L. 34.500 (Abb. L. 31.050)

Per ordinare il volume utilizzare l'apposito tagliando inserito in fondo alla rivista.

L'ESIGENZA DI CAPIRE.
DICTIONARY OF COMPUTER SCIENCE
English Italian German Italian-English German-English
DIZIONARIO DI INFORMATICA
Inglese-Italiano-Tedesco Italiano-Inglese Tedesco-Inglese
WÖRTERBUCH DER INFORMATIK
Englisch Italienisch Deutsch Italienisch-Englisch Deutsch Englisch
Otto Vollnhals
GRUPPO EDITORIALE JACKSON
15.000 TERMINI
UNA PROPOSTA DEL GRUPPO EDITORIALE JACKSON
Pag. 920 Formato 18 x 25 cm. Cod. 100 H L. 45.000 (Abb. L. 36.000)
Per ordinare il volume utilizzare l'apposito tagliando inserito in fondo alla rivista.

CEDOLA DI COMMISSIONE LIBRARIA

Da inviare a Gruppo Editoriale Jackson - Via Rosellini, 12 - 20124 Milano

Nome Cognome

Indirizzo

Cap. **Città** **Provincia**

Partita I.V.A. (indispensabile per le aziende)

Si richiede l'emissione della fattura □

Inviatemi i seguenti libri:

Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità

□ Pagherò al postino il prezzo indicato + L. 2.000 per contributo fisso spese di spedizione.

□ Allego assegno n° .... di L. ....

□ Non abbonato □ Abbonato sconto 20% □ l'Elettronica □ Elettronica Oggi □ Automazione Oggi □ Elektor
□ Informatica Oggi □ Computerworld □ Bit □ Personal Software □ Strumenti Musicali □ Videogiochi

Data .... Firma ....

CEDOLA DI COMMISSIONE LIBRARIA

Da inviare a Gruppo Editoriale Jackson - Via Rosellini, 12 - 20124 Milano

Nome Cognome

Indirizzo

Cap. **Città** **Provincia**

Partita I.V.A. (indispensabile per le aziende)

Si richiede l'emissione della fattura □

Inviatemi i seguenti libri:

Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità

□ Pagherò al postino il prezzo indicato + L. 2.000 per contributo fisso spese di spedizione.

□ Allego assegno n° .... di L. ....

□ Non abbonato □ Abbonato sconto 20% □ l'Elettronica □ Elettronica Oggi □ Automazione Oggi □ Elektor
□ Informatica Oggi □ Computerworld □ Bit □ Personal Software □ Strumenti Musicali □ Videogiochi

Data .... Firma ....

L. 16.000

Cod. 336 D ISBN 88-7056-133-X

Questo libro è un validissimo supporto e strumento di "lavoro" per tutti coloro che vogliono o devono imparare a programmare in BASIC con un personal computer Commodore o con l'M20 Olivetti.
In modo schematico ma rigoroso, completo ma semplice, e sempre con l'ausilio di numerosissimi esempi ed esercizi, il lettore potrà trovare, tutte le istruzioni, comandi, informazioni e consigli che gli permetteranno, in modo rapido ed approfondito, di programmare in BASIC.
Dopo un'introduzione di carattere generale (sistema binario e struttura base di programma...) si entra nel vivo della programmazione.
Si susseguono: costanti e variabili, variabili array, i comandi, le istruzioni, la preparazione di un flow-chart, il trattamento delle stringhe, la stampante, alcune routine di gestione stampa, i nastri, il disco, comandi di DOS support, i file dati, i messaggi d'errore, la tecnica di funzionamento overlay,...
Vi sono infine alcune Appendici tra le quali troverete le funzioni speciali del CBM 8000 e alcuni programmi dimostrati su M20.
In un solo libro, insomma, è compendiato tutto quello che occorre per utilizzare al meglio due tra i più diffusi Personal Computer.

78

IL BASIC DEL PET E DELL'M20

Paolo Pascolo e Carlo Pascolo

GRUPPO EDITORIALE JACKSON